Universidad
Carlos III de Madrid

Trabajo Fin de Grado

Desarrollo de una aplicación de localización GPS por Realidad Aumentada para dispositivos Android

Grado en Ingeniería Electrónica Industrial y Automática

Autor: Eduardo Alvarado Piñero

Tutor: Ángel García Crespo

(Hoja en blanco)

Agradecimientos

Han sido 4 años de mucho afán y dedicación. A día de hoy y por diversas situaciones, un estudio universitario se merece muchísimo tiempo y esfuerzo para conseguirlo, por parte del estudiante y, en muchas ocasiones, de la familia y amigos.

Este proyecto no ha sido solo mío, sino de mis padres y abuela, que desde que entré por las puertas de la universidad estuvieron dándome todo su apoyo y confianza día tras día; Mi hermano, que durante estos 4 años me apoyó, aconsejó y asesoró para que mi estancia fuera lo más fructífera y feliz posible; y mis amigos, que hicieron que el aprendizaje de casi un lustro no solo fuera académico, sino que también fuera sobre la vida.

Muchas gracias a todos.

(Hoja en blanco)

Resumen

El siguiente proyecto consiste en la planificación, desarrollo y ejecución de una aplicación o programa escrito en lenguaje JAVA bajo el entorno Android, sistema operativo para dispositivos móviles.

Esta aplicación se basa en la idea de geolocalización mediante GPS y el uso de la Realidad Aumentada, para mostrar en pantalla distintos puntos de interés a través de la cámara, que, en tiempo real, cambian de tamaño y se mueven en función de la posición del usuario.

Para llevar a cabo este proyecto, se hará uso del conocimiento en lenguaje JAVA y Android, así como Eclipse en cuanto al entorno de programación. Y además, con el uso de una cuenta gratuita de desarrollador de Google, se tendrá la capacidad de usar distintas API's con el objetivo de mejorar nuestro programa (por ejemplo, incluyendo un mapa provisto por Google Maps).

Para la Realidad Aumentada, se hará uso de las librerías BeyondAR, de carácter Open Source y totalmente gratuitas. Nos ofrecerán las herramientas necesarias para superponer los objetos virtuales sobre la imagen de la cámara, y sobre el mapa.

Todo ello se explicará detalladamente paso por paso, desde la construcción del entorno de desarrollo, hasta la creación de la aplicación final, pasando por la descripción en detalle de Android y su arquitectura, o las librerías de BeyondAR y sus funcionalidades.

Palabras clave

Realidad Aumentada, Realidad Virtual, GPS, Geolocalización, localización, Android, Eclipse, Java, BeyondAR, World, GeoObject, LocationListener, onLocationChanged.

Abstract

The following project consists in the planning, development and execution of an application or program written in JAVA language under the Android environment, an operating system for mobile devices.

This application is based on the idea of using GPS Geolocation and using Augmented Reality, to display several points of interest through the display, which, in real time, change size and move as a function of the position user.

To carry out this project, we will make use of knowledge in JAVA language and Android, as well as the Eclipse programming environment. And with the use of a free Google Developer account, we will have the capacity of using different API's in order to improve our program (for example, including a map provided by Google Maps).

For the Augmented Reality, we will use BeyondAR libraries, Open Source and totally free. They will provide the necessary to overlap virtual objects on the camera image, and map.

All this is explained in detail step by step, from the construction of the development environment to the creation of the final application, through the detailed description of Android and its architecture, or BeyondAR libraries and their functionalities.

Keywords

Augmented Reality, Virtual Reality, GPS, Geolocation, Location, Android, Eclipse, Java, BeyondAR, World, GeoObject, LocationListener, onLocationChanged

(Hoja en blanco)

Índice general

Índice de figuras

Índice de Tablas

(Hoja en blanco)

1. Introduccion y Objetivos

1.1. Introduccion

Aun todos recordamos esa escena en la pelicula Terminator 2: El dia del Juicio Final, que cambió nuestra perspectiva sobre el futuro de la tecnologia. Era de las primeras veces que se nos presentaba una posibilidad asi, que tan ficticia parecia entonces, y tan factible podemos encontrar a dia de hoy.

Nuestro personaje principal, aparentemente indefenso, se dispone a entrar en un bar de carretera atestado de moteros y otra gente no dispuesta a hacer muchos amigos. Desde fuera, parece una persona normal y corriente, pero cuando entra en el recinto empieza a escanear a cada uno de los integrantes. Desde el punto de vista de los ojos del personaje, durante la escena se nos muestra lo que este ve en todo momento, mientras lineas de codigo, numeros e informacion llenan la vista del Terminator sobre un atractivo y futurista filtro de color rojo, como si del monitor de un ordenador de alta tecnologia se tratase.

Es un **constante analisis del entorno**. Con cada persona que se encuentra, es capaz de extraer un conocimiento que de otra forma no podria haber sabido. Su nombre, que ropa lleva y si es de su talla o incluso si representa una amenaza en funcion de los detalles citados.

Desde ese momento, toda una generacion de cinefilos y no tan cinefilos les habria gustado, al menos por un segundo, imitiar a Arnold Schwarzenegger mientras caminaba por la calle. 23 años han pasado desde entonces, y quizas la impresión futurista que nos ofrece esa escena nos siga dibujando una sonrisa en la cara, pero tambien es cierto que lo que antes parecia un sueño, a dia de hoy se ha convertido en una aspiracion altamente realizable.

Quizás mientras grababa James Cameron la famosa escena, pensaba si algún día la realidad superaría a la ficción. Puede que aún sea pronto para saberlo. Pero si de algo que podemos estar seguros es, a ciencia cierta, de que en este ámbito el mundo esta cambiando. Y no hace falta irse 23 años atrás para percartarse de que estamos en constante evolución.

Como humanos nos hemos caracterizado siempre a lo largo de la historia por propiedades que aún nos siguen asombrando a nosotros mismos. Ha sido nuestro afán por el hallazgo y el progreso el que nos ha hecho descubrir, idear y concebir todas las mejoras y avances que hasta día de hoy, nos rodean y nos han encumbrado como especie.

Es ese momento en el que no nos conformamos con nuestra naturaleza y con lo que vivimos; con lo que vemos y oímos, cuando nos damos cuenta de que todas esas propiedades que nos hacen ser lo que somos, también nos pueden ayudan a ser lo que no somos, y James Cameron fue uno de los que se dio cuenta de ello a la hora de escribir el guion de su película. ¿Por qué no ver más allá de lo que vemos? ¿O por qué no oír más allá de lo que oímos?

Cuanto más nos adentramos en una época donde la comunicación es el pilar fundamental de la cultura actual, más sentido cobran estas preguntas, que, como si de elementos de una película se tratasen, nos dan cualidades especiales para entender, divagar y adentrarse más

aun en esta edad de la información. Ha sido esa unión entre el individuo y el mundo informático la que se ha estado buscando durante todo ese tiempo. Esa simbiosis perfecta que nos permita ser un pez más en un enorme océano de bits. Que destruya barreras y nos haga recibir a la tecnología con los brazos abiertos.

Y los "trampolines" que nos permiten saltar a ese mar de ceros y unos son más, y cada vez más variados. Desde las más básicas redes sociales, mensajería, etc., hasta televisiones inteligentes cuyas aplicaciones nos permiten realizar, desde videoconferencias entre distintas partes del globo, hasta hacer ejercicio con un monitor personal virtual; o videoconsolas que nos posibilitan trasmitir en directo nuestras partidas, mientras jugamos con otros jugadores que no conocemos en persona. Pero el personaje principal de Terminator 2 disponía de otro medio bastante especial.

Podríamos resumir la Realidad Aumentada como la función de añadir información a lo que sabemos, o simplemente a lo que queremos saber y desconocemos. La realidad aumentada nos permite poder ver de forma ampliada en información lo que sentimos a nuestro alrededor.

Si miramos al cielo de noche, veremos una bonita bóveda oscura repleta de estrellas que, según nos contaron en su momento, forman curiosas y variadas formas llamadas constelaciones. Aun así, solo un experto en la materia podrá decirte el nombre de cada una y su posición. Al menos antes. ¿Por qué no mirar al cielo, y además de un bello paisaje nocturno, no poder ser capaz de determinar qué es lo que vemos? Los nombres, las posiciones de los planetas, o de las estrellas. Ampliar lo que somos capaces de observar. Sumar y sobreponer información que nos ayude a comprender sin tener previo conocimiento. Google lo logró son Sky Map, pero esa no es más que una de las muchas posibilidades que se nos presentan en este mundo repleto de oportunidades.

Con este proyecto no pretenderemos alcanzar el nivel de Arnold en su película, sino que mostraremos que el camino que sugirió James Cameron está ahí. La Realidad Aumentada es un hecho, existe, y a cada día que pasa, esta tecnología da un paso más hacia lo que nos gustaría que fuese; hacia lo que vemos en las películas.

1.2. Objetivos

Con este proyecto, pretendemos realizar una aplicación para dispositivos móviles bajo el sistema operativo Android, haciendo uso del a cámara del teléfono y de una serie de librerías de Realidad Aumentada (BeyondAR).

Posteriormente, se hará uso de esa aplicación en un hardware "vestible", como son unas gafas de realidad aumentada.

El objetivo **principal** de esta aplicación será:

- Desarrollo de una aplicación de realidad aumentada, en la que introduciendo una ruta (latitud y longitud), seremos capaces de mostrar en pantalla el punto de interés por geolocalización. El lugar del punto en la pantalla variará dependiendo de la posición a la que estemos mirando, y de la distancia a la que estemos de ese punto.

Esta aplicación se cargará en unas gafas capacitadas[1] especialmente para el caso.

Los objetivos **secundarios** serán:

- Estudiar desde cero a programar en lenguaje Java, ya que Android hace uso de este lenguaje.
- Aprender a usar como entorno de desarrollo integrado (IDE) Eclipse y sus diversas funciones para el desarrollo.
- Una vez con conocimientos en Java, comprender y aprender el funcionamiento y arquitectura del sistema operativo Android, así como sus herramientas y especializarse en ellas (entre otras, por ejemplo el lenguaje de marcas XML).
- Entendimiento de un sistema de localización por GPS, su funcionamiento y su programación.
- Manejar y saber utilizar librerías de Realidad Aumentada y tener un primer contacto con esta tecnología.

1.3. Motivación

La implementación en la sociedad actual de una idea semejante resultaría en un cambio y una nueva concepción de lo que conocemos como comunicación a día de hoy. El poder conectar de una forma tan potente con la información podría tener consecuencias increíblemente ventajosas. Superponer información que no vemos a simple vista facilitaría las tareas de innumerables personas, tanto en el aspecto profesional, como el de ingenieros, hasta en un ámbito más del entretenimiento, como en cine o videojuegos. Y por supuesto en un ámbito médico, en el que tanto profesionales en sus entornos de trabajo (i.e. operaciones quirúrgicas) como personas con discapacidades (i.e. con problemas de visión) harían uso de todas estas ventajas.

Además, otra motivación es sin duda el conocimiento en programación Java y Android.

Referente a Java, es uno de los lenguajes líderes de la industria a día de hoy. Ofrece gran versatilidad, ya que puede ser usado en un gran abanico de dispositivos. También es independiente: Java puede funcionar en diversos sistemas operativos (Apple's OS X, versiones de Linux, Microsoft Windows…).

Una cualidad vital es que está Orientado a Objetos. Esto nos permite una inmensa eficacia a la hora de programar. Y además, es moderno. Un lenguaje actual y con gran perspectiva de futuro.

En cuanto a Android, estamos ante un sistema operativo basado en Linux, desarrollado por Google y la **Open Handset Alliance**[2]. Diseñado para sistemas móviles y embebidos (desde móviles, tablets, hasta recientemente sistemas vestibles como relojes o gafas inteligentes) es un sistema que está en constante auge y evolución. Con un **45.01%** [3] del mercado de móviles

[1] Gafas de Realidad Aumentada EPSON Moveiro BT-200
[2] http://www.openhandsetalliance.com/
[3] Cuota de mercado mundial en Agosto de 2014, según NetMarketShare.

mundial, la cuota de mercado sigue en crecimiento con tendencia ascendente y emparejada con iOS, y todo apunta que la libertad y carácter de software sin barreras seguirá siendo apoyada por una gran parte de los usuarios en el mundo de los Smartphones.

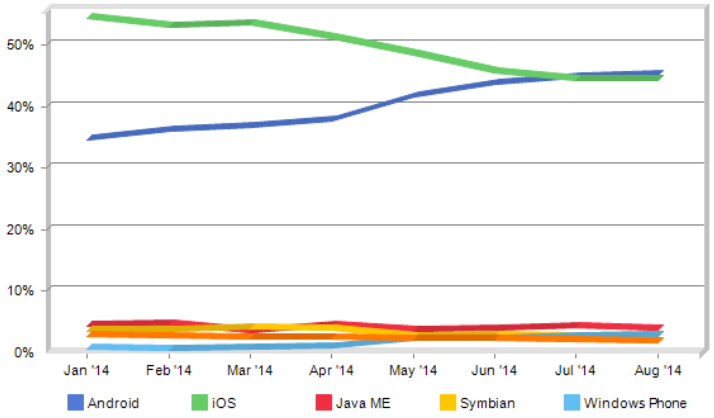


Ilustración 1: Cuota de mercado mundial de sistemas operativos para móviles. (FUENTE: netmarketshare.com/)

Mes	Android	iOS
Mayo, 2014	41.58%	48.34%
June, 2014	43.75%	45.61%
Julio, 2014	44.62%	44.19%
Agosto, 2014	45.01%	44.34%

Tabla 1: Distribución SO móviles en el mundo

Es una oportunidad única para desarrollar en Android. Este sistema nos ofrece un sólido SDK sobre el que trabajar, con un espectro de posibilidades prácticamente limitado solo por nuestra imaginación, y su naturaleza de código libre implica una comunidad de usuarios que recorre todo el globo. Y sobre todo, es barato. No es necesario una gran inversión (y en ocasiones ni siquiera eso), para empezar a desarrollar.

1.4. Fases de desarrollo

- **Fase 1: Introducción al lenguaje de programación JAVA.**

A la hora de empezar con la realización del proyecto, la gran dificultad fue estudiar desde cero un lenguaje que hasta ese momento me era desconocido, puesto que siempre había programado en C y C++. Por lo que el primer paso fue la adecuación a este lenguaje, a su sintaxis y sus conceptos principales, como la programación orientada a objetos (polimorfismo, herencia, encapsulamiento, abstracción…). Una vez con unos conocimientos básicos, el aprendizaje se amplió hacia aspectos y características más concretas del JAVA. Todo ello se dió gracias a la documentación del libro “Aprenda Java como si estuviera en primero” (Garcia de Jalón, y otros, 2000)

- **Fase 2: Adecuación del entorno de desarrollo.**

La segunda fase fue la construcción y puesta a punto de nuestro entorno de trabajo para la realización de aplicaciones Android. Aunque para la realización de la primera fase haga falta tener nuestro IDE a punto para programar en Java y ambas fases se den al mismo tiempo, consideraremos está en segundo lugar por la amplia configuración requerida para realizar aplicaciones Android.

La instalación, que se detallará detenidamente más adelante, consta de la preparación de nuestro entorno de trabajo Eclipse for Java Developers, en el que incluiremos nuestro ADK (Android Development Kit) y las Android SDT (Software Development Tools), todo ello ofrecido por Google y preparado para una iniciación rápida y sencilla.

Para la adaptación de nuestras aplicaciones a la Realidad Aumentada, bastará con copiar las librerías que nos otorga BeyondAR a nuestra aplicación. No deberemos llevar a cabo ninguna configuración especial.

- **Fase 3: Documentación para programación de aplicaciones Android**

La programación para aplicaciones Android, a pesar de ser JAVA, adquiere nuevas funcionalidades y cualidades a nuestro código. Partiendo de los nuevos comandos, hasta nuestros archivos .XML que nos dictan como será y se compondrá la aplicación en la pantalla.

El aprendizaje de aplicaciones Android, se ha hecho fundamentalmente con el curso online “Android Complete!” (Tushinsky), además de diversos tutoriales y páginas web.

- **Fase 4: Realización de aplicaciones Android**

Antes de ponernos a desarrollar la aplicación que tenemos como objetivo, se realizó previamente una serie de aplicaciones que ayudarían a la comprensión de los aspectos más fundamentales, y con el tiempo, más avanzados sobre Android.

Tocando cada aspecto de este sistema operativo, se han realizado aplicaciones para ver las funcionalidades de las interfaz (botones, imágenes booleanos…), los distintos layouts y como manipularlos, aplicaciones con distintas actividades, manejo de strings (Juego del Ahorcado) y un largo etc. Todo con ayuda del también muy interesante curso de (Gómez Oliver).

El principal objetivo de esta fase fue que mi primera aplicación Android no fuera la que tenemos como objetivo en este proyecto, sino que lleváramos cierta experiencia a la hora de su realización.

- **Fase 5: Desarrollo de fases previas de la Aplicación final**

Una vez nuestro entorno de trabajo estaba listo y habíamos tenido suficiente experiencia con JAVA y el desarrollo de aplicaciones en Android, el siguiente paso era empezar a estudiar y divagar como crearíamos la solución a nuestro problema. Gracias a los tutoriales que nos ofrecía BeyondAR, el objetivo de la fase actual fue el familiarizarse con estas librerías y llevar a cabo una aplicación modelo de lo que sería nuestra solución final.

Principalmente, esta investigación tomo tres rumbos: La puesta en escena de los objetos de Realidad Aumentada en la pantalla, que se llevó a cabo en la fase que nos ocupa, la implementación del API de Google, que nos serviría para disponer de su servicio de Google Maps, y la creación de un servicio de localización propio.

- **Fase 6: Creación de un GPS.**

A pesar de que las librerías de BeyondAR disponían de un servicio de localización, se decidió el desarrollo de nuestro propio servicio para entender y comprender lo máximo posible el funcionamiento de nuestra aplicación, al ser una parte vital de esta, de cara a futuras mejoras.

- **Fase 7: Servicios de Google.**

Las fases 5, 6 y 7 fueron llevadas aproximadamente al mismo tiempo, ya que su implementación debía venir casi de la mano. En la actual fase, entro en juego la investigación de las herramientas o servicios de Google, que nos ofrecen posibilidades hasta ahora desconocidas por mi parte, como el uso de sus sistemas de mapas, gracias a la API Key que nos otorgan para nuestra aplicación sin coste alguno. Gracias a Google Developers el trabajo ha sido más sencillo.

- **Fase 8: Adecuación de la Aplicación final.**

Creación de la aplicación final. Búsqueda de la información sobre la librería de Realidad Aumentada y desarrollo.

- **Fase 9: Últimos retoques.**

Últimos detalles de la aplicación. Resolución de bugs y errores.

- **Fase 10: Testeo y prueba de errores definitiva.**

Puesta a punto, compilación y debug de la aplicación. Comprobar que todo funciona correctamente.

- **Fase 11: Redacción de la memoria.**

Una vez tenemos todo lo necesario procederemos a la redacción del guión.

1.5. Medios empleados

Los distintos medios usados en la realización de este proyecto podemos nombrarlos como:

- **Hardware**:

Ordenador de Sobremesa Intel® Core ™ i7-2600k CPU @ 3.4GHz - 3.7 GHz, 4GB RAM.
Teléfono móvil Samsung Galaxy S2 i9100 1GB
Teléfono móvil Sony Xperia Z2 3GB
Cable USB.

- **Software**:

Ordenador de Sobremesa: Windows 7 x64 bits
Teléfono móvil: Android version 4.2.2 (Jelly Bean), rom personalizada Cyanogemod 10.1.
Java 7 Update 51 (64-bit)
Java SE Development Kit 7 Update 51 (64-bit)
Eclipse IDE for Java Developers Version: Kepler Service Release 2
BeyondAR Framework
Google Maps
Google Earth
Aplicación Simulador de rutas GPS, por Fernando F. Gallego[4]

- **Otros**:

Conexión a Internet.
Impresora.

1.6. Estructura de la memoria

Con esta memoria, abarcaremos los siguientes temas. Empezando desde el Planteamiento del Problema, haremos un Análisis de los campos que se centran en el Proyecto, empezando desde una perspectiva general para acabar con una más específica y orientada a nuestro objetivo.

Luego abordaremos el Diseño de la Solución Técnica, donde explicaremos el proceso seguido a la hora de desarrollar nuestra aplicación, y posteriormente un capitulo Resultados y Evaluación expondrá como usar y comprender la aplicación desde la perspectiva del usuario final.

Para acabar, introduciremos unas Futuras Líneas del Proyecto de cara a la evolución del programa, y luego un Presupuesto, además de una Evaluación, Resultado y Conclusiones del trabajo.

[4] Google Play: Simulador rutas GPS

2. Planteamiento del problema

2.1. Análisis del estado del arte

La meta central de este proyecto es una, la Realidad Aumentada, y como tal, es preciso conocer y entender de lo que es capaz esta tecnología. Apoyándonos sobre el estudio de (Fernández Santiago, González Gutiérrez, & Remis García), podremos definir la Realidad Aumentada como la superposición de información virtual sobre información real, de tal manera que obtenemos una mezcla de datos procedentes de "ambos mundos", una realidad mixta.

2.1.1. Historia de la Realidad Aumentada

Fue en los años 50's y 60's cuando la gente empezó a comprender el concepto de Realidad Aumentada, sobre todo con la innovadora creación de un fotógrafo llamado Morton Heilig en 1962. Este invento, al que llamo Sensorama, permitía al usuario introducirse en una cabina con forma de "U" la cual simulaba ser conductor de motocicleta. Esta máquina era capaz de reproducir imagen, sonido, vibraciones, e incluso aromas. Simplificando y optimizando el concepto, Ivan Sutherland invento en 1966 el HMD, el primer display de cabeza.

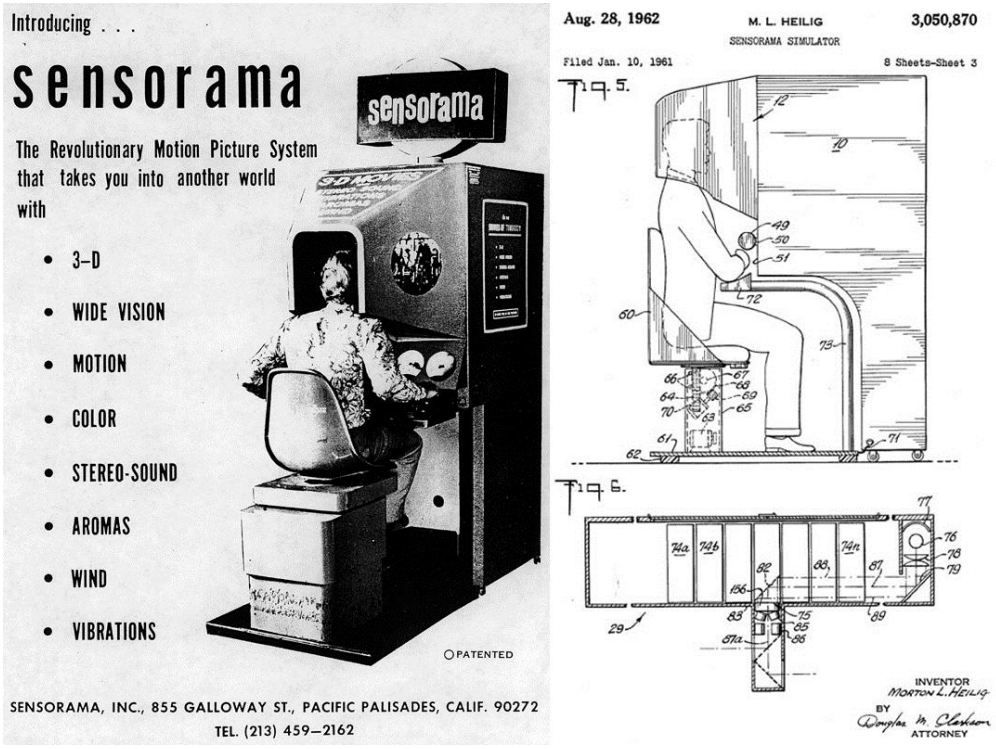


Ilustración 2: Sensorama, por Morton Heilig (FUENTE: tallera3bandas.blogspot.com.es)

El camino se definía poco a poco, paso a paso, y la interacción virtual llego finalmente en 1975 cuando Myron Krueger creo el "Videoplace", un laboratorio de realidad virtual cuyo sistema permitía interactuar con objetos virtuales.

A pesar de su avance imparable, el término de RA seguía sin ser muy popular, faltaba darlo a conocer. Esto se solucionó cuando Jaron Lanier, un informático procedente de la Nueva York de los 80's creo el VPL Research[5] y popularizo el término.

En los años 90's, tres ingenieros, Steven Feiner, Blair MacIntyre y Doree Seligmann diseñaron KARMA, un prototipo de Realidad Aumentada. Y en 1999, el japonés Hirokazu Kato desarrolló ARToolKit, mediante el HitLab.

Pero entramos en la era de la información. Años 2000's. Tras los avances de los sistemas informáticos llega el auténtico "big-bam" de la realidad aumentada. En el primer año del nuevo milenio se presenta ARQuake, el primer juego al aire libre para dispositivos móviles de Realidad Aumentada, desarrollado por Bruce H. Thomas.

Ilustración 2.2: ARQuake, desarrollado por Bruce H. Thomas (FUENTE: chopsueyblog.wordpress.com)

Los pasos de gigante tienen una cabida descomunal, y solamente 8 años después ya encontramos una aplicación, Wikitude Guía, que permite informarnos de turismo gracias a geo posicionamiento, brújula digital, contenido de la Wikipedia, etc., todo en un ámbito de Realidad Aumentada sobre plataformas Android. La era de la Realidad Virtual y Aumentada está en todo su esplendor.

2.1.2. Conceptos de Realidad Aumentada

La Realidad Aumentada no tiene una definición propiamente dicha, sino que depende del ámbito al que se aplique.

Podemos decir que la tecnología AR es un sistema cuyo objetivo es potenciar nuestros sentidos. Una lupa, por ejemplo, podría ser un elemento AR (incrementa nuestra visión), un audífono también podría serlo, ya que hace que personas con problemas auditivos escuchen mejor. La diferencia entre estos elementos y la AR que estudiaremos, es que esta última incluye elementos virtuales.

Otra definición de AR, es la modificación de la percepción que tiene el usuario en cuanto a lo que le rodea, ya sea mediante la inclusión/eliminación de objetos virtuales o reales que no estén/estén, respectivamente, en la vida real.

[5] VPL Research, primera empresa centrada en la realidad virtual.

Pero la definición más técnica y adaptada a los tiempos que corren sobre esta tecnología viene dada por Ronald Azuma (Azuma, August 1997). Según el ingeniero japonés, la Realidad Aumentada es un entorno que incluye elementos de Realidad Virtual y elementos del mundo real. Como ejemplo, propone una persona que lleva unas gafas a través de las cuales es capaz de ver el mundo, y superponiéndolas, imágenes generadas por ordenador proyectadas sobre dicho mundo.

Por tanto, de esta descripción podemos extraer 3 características o conceptos fundamentales que compondrán nuestras actuales y futuras aplicaciones Android en este aspecto:

- Es un sistema en 3 Dimensiones.
- Es interactivo en tiempo real.
- Es una combinación de mundo real y mundo virtual.

2.1.2.1. Realidad Aumentada vs Realidad Virtual

Es común la confusión que se produce al hablar de ambos términos, relacionados, pero que albergan diferencias importantes.

Cuando hablamos de realidad virtual, según Jaron Lanier, fundador de "VPL Research" como vimos antes (2.1.1), es un entorno generado por ordenador, interactivo, tridimensional en el cual se introduce al individuo.

Tal y como hicimos antes con la tecnología AR, podemos sacar 3 características importantes de esta definición de Realidad Virtual:

- El entorno virtual esta generado por ordenador en un ámbito de 3 Dimensiones, el cual requiere cierta potencia gráfica.
- Es interactivo en tiempo real, siendo totalmente necesario este punto para que la interacción usuario-entorno se haga de manera efectiva.
- El usuario está inmerso en el mundo virtual.

Si comparamos estas características con las que se describían en la tecnología AR, solamente poseen como gran diferencia el tercer punto, el nivel de inmersión.

En un sistema virtual, la persona está totalmente cubierta por el mundo creado por ordenador. Sin embargo, con la Realidad Aumentada, esta se encarga de "ampliar" el mundo ya existente. El primero lo sustituye, el segundo lo complementa.

Esto hace que en un sistema de Realidad Virtual, se imposibilite al usuario interactuar con el mundo real. Por otro lado, con un sistema AR la interacción con el mundo real sigue siendo imprescindible.

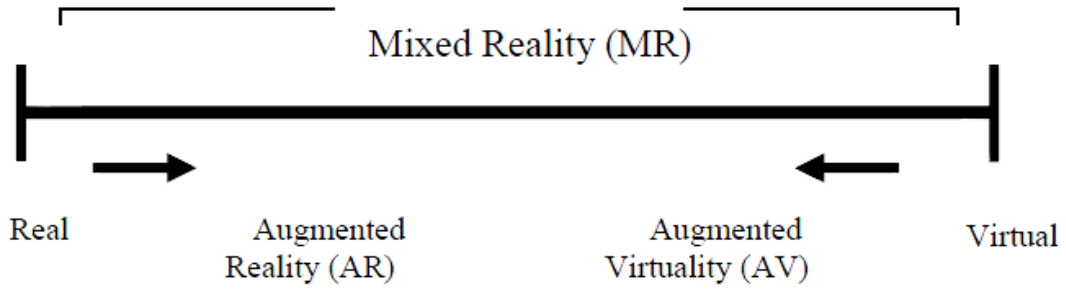


Ilustración 3: Realidad Aumentada vs. Realidad virtual (FUENTE: Universidad de Oviedo)

2.1.3. Tecnología de Realidad Aumentada

Para todo sistema AR, será necesario una serie de componentes: Un dispositivo que se encargue de captar la información del mundo que le rodea en tiempo real. Una máquina, capaz de crear imágenes virtuales y procesarlas para sobreponerlas a la información real recogida por el dispositivo, y un medio, que proyecte el resultado final.

De tal forma que deben de tener siempre estos componentes, podemos encontrar distintas maneras de llevarlas a cabo.

2.1.3.1. Soluciones Hardware

Los sistemas de Realidad Aumentada requieren cierto tipo de arquitecturas, específicas y cómodas, que conlleven un óptimo uso de esta ciencia. Al ser una tecnología cuyo objetivo es la aportación en tiempo real de información, es recomendable que este dirigida había tecnologías portátiles, ligeras, de bajo peso, fáciles de usar e intuitivas. Como una de las grandes ventajas de la Realidad Aumentada reside en la movilidad, disponer de elementos de gran tamaño reduciría considerablemente los puntos fuertes de esta tecnología.

2.1.3.1.1. Arquitecturas de cabeza

Claramente, la mejor forma de poder transportar algo es sin el uso de las manos, y que mejor lugar que la cabeza, sobre todo cuando la Realidad Aumentada se centra primordialmente en la vista.

Los HMD [6](Head-mounted Display) son dispositivos capaces de acoplarse a la cabeza del usuario, de la misma manera que hace un casco. Su posición facilita mucho las tareas llevadas a cabo, principalmente por dos causas.

En aplicaciones AR visuales, la entrada de datos del mundo real siempre se dará lugar a través del ojo, y unos dispositivos a la altura de la mirada ayudarían a percibir dicha entrada con mucha más facilidad, además de poder orientar la adquisición de datos de manera intuitiva.

[6] Casco de Realidad Virtual/Realidad Aumentada

Por ejemplo, si queremos saber dónde se encuentra la salida del metro, miramos hacia donde creemos que esta, y la aplicación nos muestra con su hardware su posición exacta superponiendo la información, de manera que todo es instantáneo.

La otra gran ventaja es la proyección del resultado final. No habrá esfuerzo alguno para observar la Realidad Aumentada, ya que, literalmente, la tendremos enfrente.

Ilustración 4: Sony HMD HMZ-T3W (FUENTE: sony.com)

Este tipo de hardware también posee sus inconvenientes, principalmente uno. Al ser dispositivos con forma de casco o gafas, se debe tender a hacerlos lo menos voluminosos y ligeros posibles. Por lo que se ramifica en tres problemas más: Debemos concentrar una unidad de procesamiento potente en un espacio lo más pequeño posible, y el uso de materiales lo suficientemente ligeros y resistentes como para que soporten su uso cotidiano sin agotamiento por parte del usuario. Y por supuesto, debemos tener una batería lo suficientemente ligera como para que administre la energía necesaria y tenga una autonomía convincente.

Actualmente con la continua mejora de los procesadores y la microelectrónica se está avanzando mucho en el ámbito del tamaño, siendo capaces cada vez más de albergar más potencia en menos espacio.

2.1.3.1.1.1. Clasificación de HMD's según el campo de visión

Estos dispositivos podríamos clasificarlos según varios criterios. Uno de ellos, según sean monoculares o binocular.

Los HMD monoculares están destinados a la observación mediante un solo ojo. Están destinados primordialmente a la Realidad Aumentada, ya que al necesitar solamente un punto de visión, podemos seguir observando gran cantidad de material procedente del mundo real, indispensable para este tipo de aplicaciones.

Los HMD binoculares abarcan el campo de visión de ambos ojos. En estos casos, el ángulo visual del mundo real es más reducido, por lo que esta arquitectura se centra más en la Realidad Virtual, y no tanto en la Realidad Aumentada (aunque hay muchas excepciones).

2.1.3.1.1.2. Clasificación de HMD's según el sistema

Otro sistema de clasificación, y no menos importante, es debido a la tecnología usada en el sistema HMD. Aunque a medida que pasa el tiempo, son más tecnologías las que llegan a nuestros ojos (y nunca mejor dicho), las mostradas a continuación son las más importantes.

- **Basada en lentes reflectantes (Sistema óptico)**

  Los ojos del usuario miran hacia unas lentes reflejantes parcialmente (dejan pasar cierta cantidad de luz, por lo que vemos que hay al otro lado). El elemento virtual a superponer en la capa del mundo real se refleja en la lente, de forma que el usuario vera dicho elemento, y el mundo que hay detrás de la lente.

  Los primeros sistemas con esta tecnología dejaban pasar pequeñas cantidades de luz. Para mejorar su eficacia, sobretodo con monitores monocromo, se estudió dejar transmitir una variable cantidad de luz dependiendo de su longitud de onda.

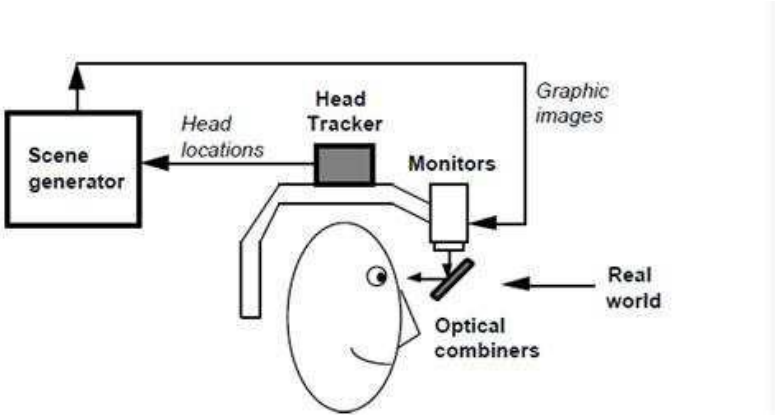


Ilustración 5: Sistema basado en lentes reflectantes (FUENTE: Universidad de Oviedo)

Es uno de los métodos utilizados en HUD's militares, como en pilotos de aviones de combate. Si nos fijamos en el esquema, este tipo de arquitectura no recoge información del mundo real (con una cámara por ejemplo), solo del usuario. Por lo que sobre la lente solo podríamos aportar información acerca de la persona que lleva puesto el HMD (altitud, orientación, posición, etc.)

Para solucionar eso, actualmente ha bastado la inclusión de un sistema de cámaras para poder examinar lo que se observa. Con ello tendríamos el concepto de gafas de Realidad Aumentada actual, las cuales son capaces de analizar información del usuario y de su alrededor.

- **Basados en cascos con monitores y videocámaras (Sistema de video)**

  En este caso, no son lentes, sino monitores los que se ponen delante de nuestros ojos, impidiendo la visión directa del mundo real. Una cámara frontal recoge la información del entorno exterior, esta se analiza, y luego el casco la combina con los elementos creados por ordenador, que se analizan previamente para conocer donde situarlos y su comportamiento.

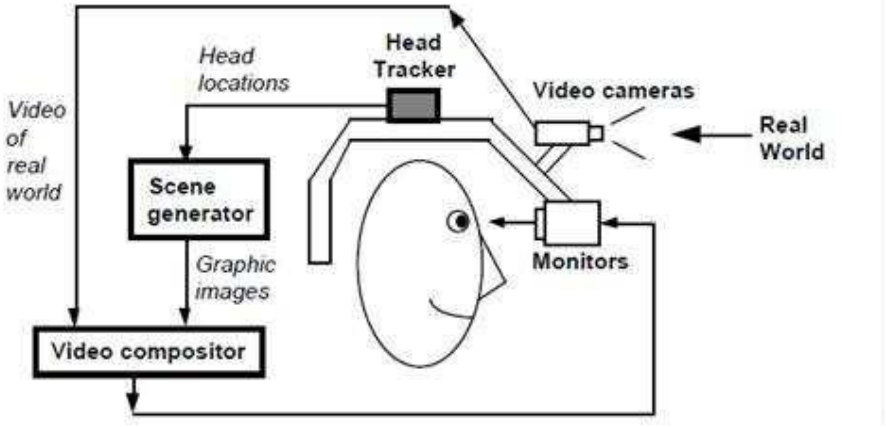


Ilustración 6: Sistema basado en monitores y cámaras (FUENTE: Universidad de Oviedo)

Esta composición de imagen se puede llevar a cabo de varias maneras. Una muy usada en los tiempos actuales es en los chroma de los efectos especiales, usados en el mundo del cine. Las imágenes generadas tendrán un fondo del mismo color que el chroma (usualmente verde), y se reemplazaran las zonas de ese color por la imagen que grabe la cámara.

Podemos darnos cuenta en esta arquitectura, que si eliminásemos el componente de la cámara, estaríamos ante un caso de Realidad virtual, el cual nos abarcaría todo el campo de visión sin ser capaz de mostrar imagen del mundo real.

- **Basados en monitores externos (Sistema de video)**

  En este caso, la cámara no se encuentra implementado en el usuario, sino que funciona en una posición separada, frecuentemente hacia ti. Es el sistema utilizado por dispositivos de entretenimiento, como la Kinect o PSEye de PS4, que nos graban al mismo tiempo que implementan información virtual.

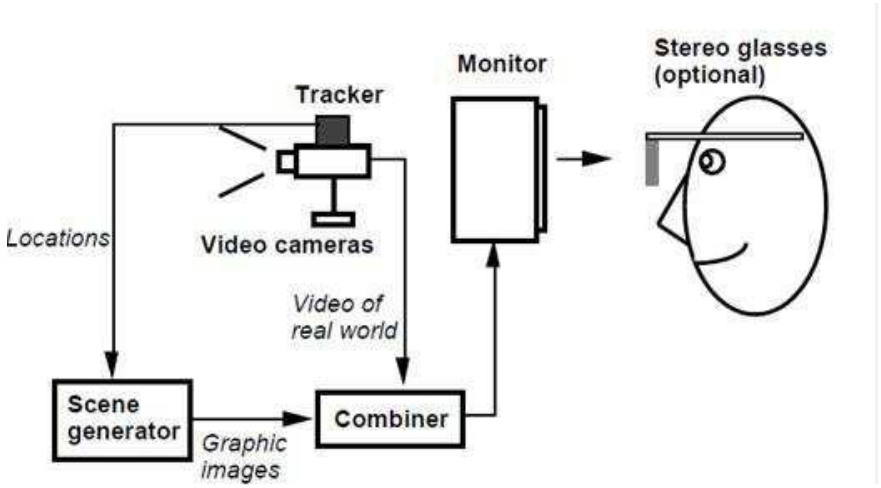


Ilustración 7: Sistemas basados en monitores externos (FUENTE: Universidad de Oviedo)

2.1.3.1.1.3. Diferencias entre sistemas ópticos y sistemas de video

Los sistemas ópticos conllevan más ventajas frente a los de video.

- Resolución y simplicidad. En un sistema de video, la captura y procesado de la imagen conlleva casi siempre una pérdida de calidad y retraso de señal. En los sistemas ópticos todo se percibe a través de las lentes, con unas perdidas menores de resolución y sin ningún tipo de retraso.
- Incapacidad. En los sistemas de video, si la imagen se apaga, el usuario no vería nada (seria como ver una televisión apagada). En los sistemas ópticos aunque no esté operativo podrá seguir viendo a través de él.
- Exactitud. En un sistema de video, la cámara debe posicionarse, en el caso de que queramos implementarla en el usuario, en una posición lo más cercana a la altura de los ojos. Pero aun así siempre conllevara cierta desviación que hará que la exactitud nunca alcance a la de un sistema óptico, que sí que están a la altura de los ojos.

En cambio, los sistemas de video también tienen ciertas ventajas frente a los ópticos:

- Realidad e inmersión: Como los sistemas ópticos deben dejar pasar la luz, los objetos virtuales no pueden ocupar todo el espacio disponible, por lo que la sensación de realidad se pierde. Es algo arreglable con los sistemas de video.
- Campo de visión. En unas gafas con un sistema óptico, la lente se sitúa a una distancia del ojo que puede deformar el objeto virtual por culpa de nuestra fisiología ocular. Los sistemas de video pueden procesar la imagen para evitar esa distorsión, haciendo que el usuario pudiera tener un campo de visión mayor que el normal.
- Control de la imagen real: En un sistema óptico, la parte real es la que es, invariable e inmodificable. En un sistema de video, podemos modificar la parte real a nuestro antojo para la finalidad que queramos, como para sincronizar ambos mundos y que no haya retraso entre ellos.

Por supuesto, con el paso del tiempo y avances tecnológicos las ventajas e inconvenientes de los sistemas varían, así como las funcionalidades.

Actualmente los sistemas ópticos montables en el usuario disponen de cámaras que son capaces de analizar la información que le rodea. Estaríamos hablando de sistemas híbridos que tendrían lo mejor de ambos sistemas, óptico y de video.

2.1.3.1.1.4. Ejemplos de monturas

- **Google Glass:** Gafas de Realidad Aumentada ligeras y compactas. Las primeras del mercado, estas gafas supusieron un fuerte contacto con la sociedad. Presentadas en Abril de 2012 por la división de Google, Google +, mostró al mundo una solución mucho más moderna y viable que los intentos anteriores. En continuo desarrollo, actualmente el objetivo es crear un sistema que se pueda acoplar a cualquier tipo de montura.

Ilustración 8: Google Glass, por Google (FUENTE: glassappsouce.com)

- **EPSON Moveiro BT-100:** Las monturas con las que trabajaremos en la realización de este proyecto. Binoculares, de lentes transparentes y cerradas, abarcando todo el campo de visión, son capaces de proyectar multitud de aplicaciones soportadas por el sistema operativo Android.

Ilustración 9: EPSON Moveiro BT-100 (FUENTE: epson.es)

2.1.3.1.2. Arquitecturas de mano

El método más sencillo y al alcance de todos a la hora de desarrollar aplicaciones de Realidad Aumentada. Si algo tenemos todos a día de hoy es un Smartphone, teléfonos móviles con conexión a internet y microprocesadores de alto nivel capaces de hacer funcionar ejemplos de tal calibre.

La mayoría de aplicaciones de AR están orientadas al uso de los teléfonos móviles, ya que obviamente los HDM aún no están adaptados a la sociedad. Dichas aplicaciones están dirigidas principalmente al seguimiento del dispositivo (o lo que es lo mismo del usuario): Geo posicionamiento GPS, sensores como acelerómetros/Giroscopios…

Es un método que viene perfecto a la hora del desarrollo temprano de ciertas aplicaciones AR por su facilidad y accesibilidad al programador.

Ilustración 10: Realidad aumentada en Smartphones (FUENTE: sync.nl)

Por otro lado, los inconvenientes que sufren las arquitecturas de mano son las limitaciones, ya bien de procesador, memoria RAM, o de resolución de pantalla (aunque estas limitaciones son cada vez menores).

2.1.3.2. Soluciones Software

A la hora de desarrollar una aplicación o un programa, necesitaremos un Software que nos permita llevar las tareas a cabo. En estos caso, detallaremos que posibilidades hemos tenido a la hora del desarrollo, tanto en el aspecto básico de programación, así como en el mundo de la Realidad Aumentada.

2.1.3.2.1. Free integrated development environment

A continuación detallamos los entornos de programación que soportan el desarrollo de aplicaciones Android, totalmente gratuitos.

- Eclipse for Java Developers: Junto al SDK (Software Development Kit) de Android, que incluye depurador de código, biblioteca, simulador de terminales, etc., y el ADT

(Android Development Tools plugin), es el entorno por excelencia no oficial para el desarrollo Android.

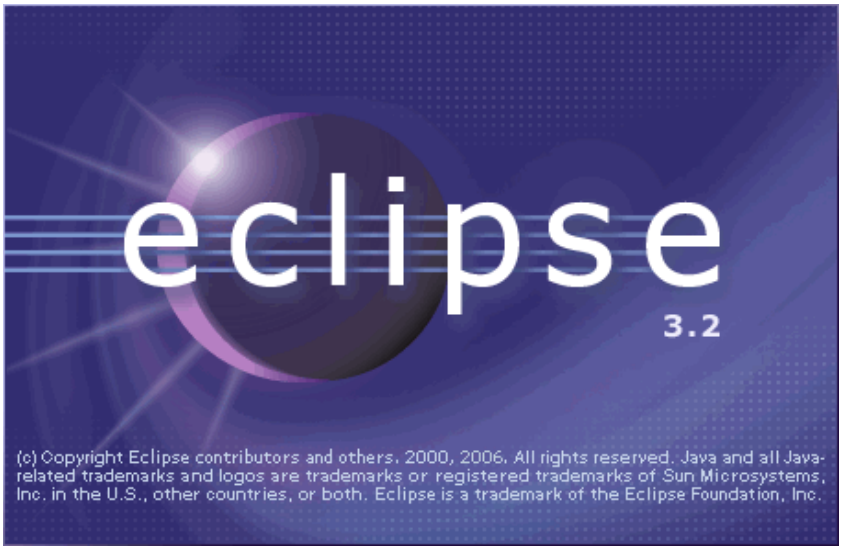


Ilustración 11: Eclipse IDE (FUENTE: eclipse.org)

- Android Studio: Un entorno nuevo y en progreso por Google (Actualmente en fase beta), basado en IntelliJ IDEA[7]. Apuntando a ser el entorno oficial para desarrollo Android, este sistema ofrece flexibilidad a la hora de construir la aplicación, plantillas para Google Services, un completo editor gráfico de .XML y un rendimiento mejorado.

2.1.3.2.2. Software AR

A la hora de introducir un framework de Realidad Aumentada, disponemos de distintas opciones, cada una con sus ventajas e inconvenientes. A continuación nombramos una de ellas, obteniendo la información de la web: (Comparacion de SDK's para Realidad Aumentada)

Como Open Source, disponemos de la que haremos uso, BeyondAR, así como DroidAR o Mixare, pues son las más importantes en torno a las AR. Como Free Commercial SDK, Layar, ARToolKit o Wikitude son entornos y librerías más profesionales y completas que el resto, que otorgan multitud de herramientas muy depuradas y listas para el uso.

2.1.3.3. Smartphone

Cuando hablamos de móviles inteligentes, los primeros que se nos vienen a la mente son los que circulan en nuestras calles a día de hoy, pero echamos la vista atrás y en ocasiones cuesta pensar cual fue el origen de semejante revolución.

Para ello, veamos primero una definición para Smartphone[8]: Podemos describirlo como un teléfono móvil, construido sobre una plataforma informativa, con capacidad de almacenamiento y procesamiento de datos semejantes al de una minicomputadora, además

[7] Intellij IDEA, Java IDE desarrollado por jetBrains.
[8] Definición de teléfono inteligente.

de una conectividad superior a la de un móvil normal. Como requisito importante y casi indispensable, construido sobre una arquitectura de pantalla táctil.

Ilustración 12: Smartphone (FUENTE: parentesis.com)

Con todos estos datos, algunos pueden pensar que el inicio de la movimiento vino de la mano de Steve Jobs y su famoso iPhone, primer teléfono multi-tactil, pero los antecedentes indican que el mundo del ordenador portátil se empezó a mover mucho antes, y ni mucho menos el concepto vino de la mano de una sola persona.

Todo empezó, curiosamente en 1992 (ABC - Primer smartphone de la historia), cuando el IBM Simon[9] se presentó en el COMDEX de Las Vegas ese mismo año. Por casi 900 dólares, disponíamos de un móvil con pantalla táctil, sin botones físicos, texto predictivo, agenda, o funciones de fax. Con sus 2 MB de ROM era capaz de incluso escribir los nombres de los contactos en la agenta mediante un teclado QWERTY. Fue posiblemente el primer Smartphone del mercado.

Ilustración 13: IBM Simon, fabricado por IBM (FUENTE: research.microsoft.com)

[9] IBM Simon, el primer Smartphone de la historia.

Posteriormente, se dio el lanzamiento de dispositivos como el Apple Newton[10], cuya popularidad acabo totalmente consumida por la aparición de las PDA's.

Algunos recordaran las terminales PDA's como puede tardar un respiro, pues el auge de Apple e iOS, así como las primeras terminales Android dieron un vuelco al mercado.

Dando un salto en el tiempo hacia el momento que nos incumbe, en 2008 salió a la vente el primer dispositivo comercial con Android, el HTC Dream.

Ilustración 14: HTC Dream (FUENTE: ebitter.es)

Trasladándonos al presente, en los Smartphones actuales podemos encontrar una serie de características principales que describen la base de cualquier teléfono inteligente:

- Acceso a internet.
- GPS
- Cámara
- Sistema operativo abierto a la instalación de aplicaciones y programas
- Reproducción de video, audio, distintos formatos de texto…

Es por ello por lo que es necesaria una unidad de procesamiento potente y una memoria más amplia de lo normal, para permitir el uso simultaneo de distintas aplicaciones.

A ello debemos sumarle una pantalla táctil de alta resolución y gran tamaño que nos facilite con claridad el uso del teléfono.

Y por último, y no por ello menos importante, un sistema de batería lo suficientemente duradero como para conseguir hacer un uso provechoso de la terminal, aunque en estos casos sea el concepto mucho más atrasado comparándolo con el resto.

Relativo a la Realidad Aumentada, los Smartphone nos permiten un gran uso de esta tecnología, y sobretodo al alcance de todos: Los displays de mano nos facilitan la libertad de movimiento y la capacidades "in crescendo", haciéndolos potenciales destinatarios para los programas AR.

[10] Apple Newton

2.1.3.4. Aplicaciones AR

En una constante evolución y auge de los sistemas AR, cada día salen nuevas aplicaciones orientadas a públicos distintos. Son tantas las posibilidades que prácticamente cada día surgen ideas sobre cómo usar esta fenomenal forma de comunicar.

Turismo: Visitas guiadas en museos, ciudades, localización de puntos de interés, geo localización, etc…llevadas a cabo con Realidad Aumentada, podrían darnos información acerca de cuadros, esculturas, monumentos…como el autor, año de realización, cualidades, así como rutas GPS, como llegar a cierto destino, y un largo etc. Para el viajero resultaría una apuesta realmente útil de cara a saber qué es lo que está viendo.

Entretenimiento: Actividades de ocio, videojuegos, películas, atracciones…Una película que se complemente con unos subtítulos virtuales, una atracción de parque donde tengamos que buscar monstruos virtuales en el lugar, o un videojuego donde se teletransporte un robot al suelo de nuestro salón son algunos de los infinitos ejemplos que podemos encontrar.

Publicidad: Una nueva faceta y realmente expectante es la del marketing. Con la realidad aumentada, estaríamos hablando de publicidad personalizada según el individuo y sus gustos. Una mujer podría solamente ver anuncios sobre artículos de belleza, o un hombre la actualidad deportiva (si así le interesa a ambos). La publicidad pasaría a ser inteligente.

Ámbito profesional: Muestra de proyectos en 3D al cliente, tutoriales 3D interactivos, guías de construcción 3D en un taller…

Militar: Análisis de entorno, HUD's en aviones de combate, miras telescópicas inteligentes…

Ilustración 15: HUD militar (FUENTE: augreality.pbworks.com)

Educación: La enseñanza se volvería mucho más fácil si el alumno viera con sus propios ojos como se construyeron las pirámides de Giza, o como circula la corriente por un circuito electrónico. Todo vale en el mundo de la Realidad Aumentada.

Esto solo es la superficie de lo que se puede conseguir, y ni mucho menos hace gala de lo que verdaderamente está en nuestra mano y en la del resto con semejantes capacidades.

2.1.4. BeyondAR

La plataforma BeyondAR será la elegida para la realización de nuestra App de realidad aumentada. Nos permitirá ver objetos virtuales a nuestro alrededor de una forma sencilla para el desarrollador.

Ilustración 16: BeyondAR (FUENTE: play.google.com)

La decisión de elegir este "framework" fue por los requisitos necesarios; Hay otras muchas plataformas de código abierto que disponían de más cualidades, como scanning de códigos para la proyección de figuras virtuales en 3D sobre las que poder observar alrededor, pero en nuestro caso lo verdaderamente importante era la geo-localización.

Gracias a nuestra clase Word conseguiremos componer la imagen recibida por la cámara con imágenes virtuales que nosotros añadiremos en un punto concreto; y el usuario se moverá a lo largo del mundo a medida que nosotros lo hacemos físicamente, pudiendo observar los objetos virtuales a nuestro alrededor.

Con la posterior inclusión de ciertos plugins o los Google Services (con una API de Google) podremos mostrar mapas dentro de la plataforma o incluir un radar que nos diga donde están nuestros objetos.

2.1.4.1. Arquitectura

Una aplicación construida bajo las librerías de BeyondAR (que podemos consultar en la siguiente fuente: (Puig Sanz)) consta de una serie de elementos imprescindibles para llevar a cabo las tareas, tanto físicos como virtuales.

El procedimiento de la aplicación es el siguiente: La cámara capta la imagen de nuestro alrededor en tiempo real, para que posteriormente, la aplicación genere un mundo virtual encima del real, superponiendo objetos virtuales en distintas latitudes y longitudes. Se sitúan en las posiciones correctas gracias a los Google Services y plugin de Google Maps que se nos proporciona.

Estos elementos, se mostraran en pantalla dependiendo de un elemento único: La posición del usuario, que como si de un objeto virtual se tratase, se sitúa en el mapa, cuya posición es dada por nuestro LocationManager, es decir, el GPS.

De esta manera los objetos variaran en tamaño y orientación según nuestra localización.

Este procedimiento se repite cada vez que el LocationManager actualiza la posición del usuario.

Ilustración 17: BeyondAR en funcionamiento (FUENTE: beyondar.com/game)

2.1.4.1.1. World

Clase: com.beyondar.android.world.World

Es el elemento "contenedor". Imaginemos que tenemos una jarra llena de caramelos. La clase World seria esa jarra, la cual llenaremos con estos caramelos (objetos virtuales), además de un caramelo especial que seremos nosotros, el usuario. Esta jarra se superpone "simbólicamente" sobre el mundo real.

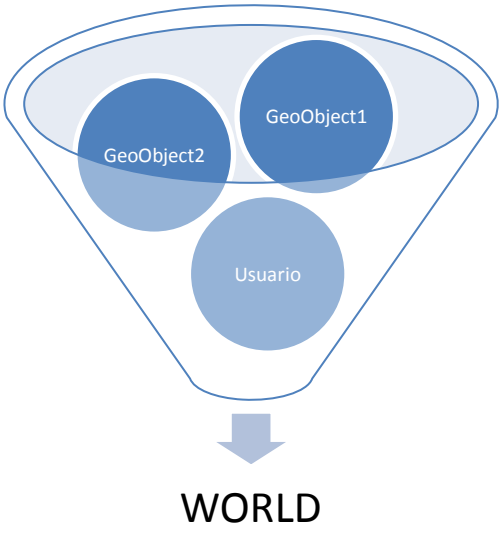


Ilustración 18: Arquitectura BeyondAR (FUENTE: Usuario)

En nuestra aplicación, crearemos una instancia sobre la que trabajaremos. Algunas de las funciones utilizadas y que nos serán de utilidad para cada instancia de la clase World:

- **synchronized void com.beyondar.android.world.World.setDefaultImage**

Cabe la posibilidad de que la imagen de nuestro objeto la obtengamos a partir de una dirección de red. En el caso de que no haya conexión y la imagen se pierda, esta función otorga una imagen predeterminada.

- **void com.beyondar.android.world.World.setGeoPosition(double latitude,double longitude,double altitude)**

Cuando configuramos una posición dentro de la clase World, lo estamos hacienda acerca del usuario, esto es, nuestro caramelo especial. En nuestro caso, esa posición vendrá dará por los datos del GPS.

- **final synchronized void com.beyondar.android.world.World.addBeyondarObject**

Si tenemos una instancia de un GeoObject (que veremos a continuación), y queremos superponerla a la clase World (siguiendo la analogía anterior, si quisiéramos meter un caramelo en la jarra), llamaremos a esta función.

- **void com.beyondar.android.world.World.addPlugin (WorldPlugin plugin)**

Usaremos esta función cuando queramos añadir un plugin a nuestra instancia de World. En este caso, el plugin utilizado servirá para mostrar los objetos virtuales sobre el mapa proporcionado por Google Services.

2.1.4.1.2. BeyondarObject

Clase: com.beyondar.android.world.BeyondarObject

Siguiendo la analogía anterior, estaríamos hablando de los caramelos que componen la jarra. Un objeto básico en el que podemos asignar una posición, entra otras muchas opciones. La extensión GeoObject (com.beyondar.android.world.GeoObject class), proveniente de BeyondarObject, hace más fácil el uso de coordenadas a la hora de situar el objeto.

- **com.beyondar.android.world.GeoObject.GeoObject (long id)**

Con esta función creamos la instancia de un solo objeto, asociado a un id para identificarlo.

- **void com.beyondar.android.world.GeoObject.setGeoPosition (double latitude, double longitude, double altitude)**

Con setGeoPosition podremos configurar la posición del objeto en el mundo virtual, latitud, longitud y altura.

- **void com.beyondar.android.world.BeyondarObject.setImageResource (int resid)**

La imagen que llevara nuestro objeto virtual, proveniente de la carpeta res.

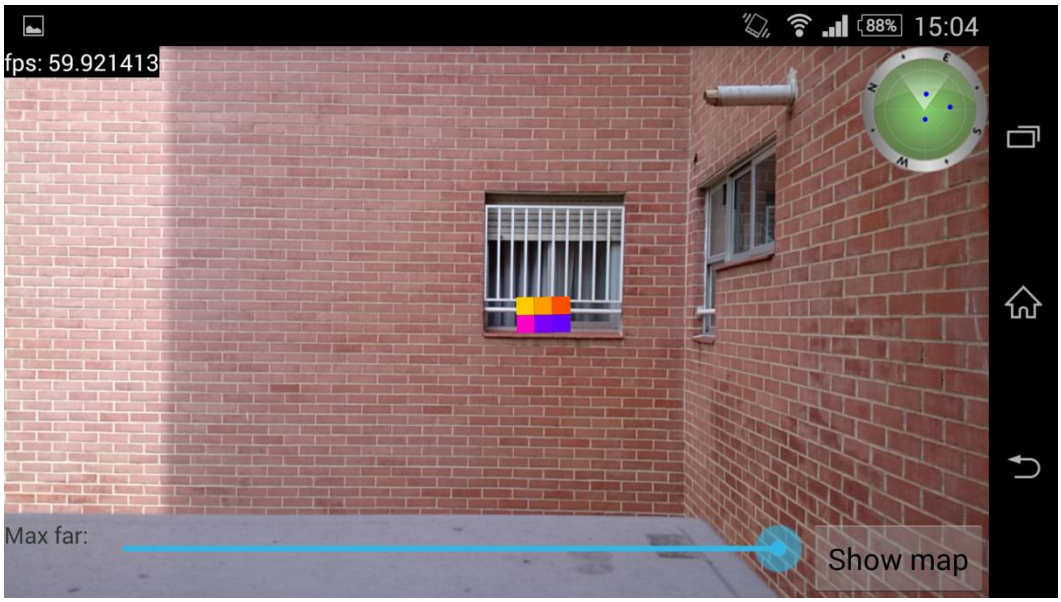
- **void com.beyondar.android.world.BeyondarObject.setName (String name)**

El nombre del objeto virtual. No se debe confundir con la identificación.

2.1.4.1.3. Distancia de renderizado

El tamaño del objeto virtual dependerá de la distancia a la que está situado del usuario, pero en ocasiones, esta distancia será tan alta que el tamaño será ínfimo, y el objeto no aparecerá.

Para ello, la arquitectura del programa nos otorgará la opción de variar la distancia de renderizado, es decir, el alcance del dibujo; cuando reduzcamos la distancia, los objetos se acercaran. El tamaño de los objetos dejará de depender de la posición, y en el momento que cambiamos la distancia de renderizado todos pasarán a tener el mismo tamaño.



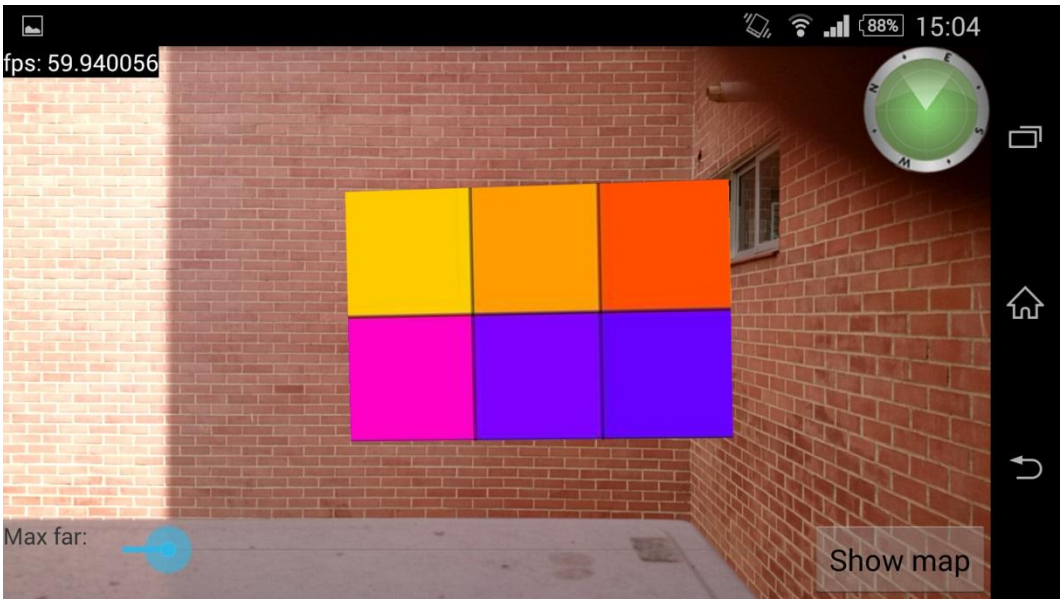


Ilustración 19: Distancia de renderizado (FUENTE: Usuario)

2.1.4.1.4. Plugin para Google Map

Clase: com.beyondar.android.plugin.googlemap.GoogleMapWorldPlugin

En nuestra aplicación tendremos fundamentalmente 2 actividades: La cámara, y todo aquello que muestra, y el mapa, provisto por Google. Cuando queramos ver nuestros objetos virtuales en el mapa, necesitaremos crear un plugin y añadirlo a la instancia de la clase World.

Para ello se crea una instancia de la clase, inicializamos, y asignamos el Mapa al que debemos configurar el plugin. Una vez tenemos el plugin configurado, lo añadimos a la instancia de la clase World [11]

```
// ...
private GoogleMap mMap;
private GoogleMapWorldPlugin mGoogleMapPlugin;
// ...

@Override
protected void onCreate(Bundle savedInstanceState) {
     // ...
     mMap = ((SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map)).getMap();

     // We create the world...
     mWorld = new World(this);
     // as we want to use GoogleMaps, we are going to create the
plugin and
     // attach it to the World
     mGoogleMapPlugin = new GoogleMapWorldPlugin(context);
     // Then we need to set the map in to the GoogleMapPlugin
     mGoogleMapPlugin.setGoogleMap(mMap);
     // Now that we have the plugin created let's add it in to our
world
```

[11] https://github.com/BeyondAR/beyondar/wiki/google-maps-plugin

```
        mWorld.addPlugin(mGoogleMapPlugin);

        // Now we fill the world
        // ...
}
```

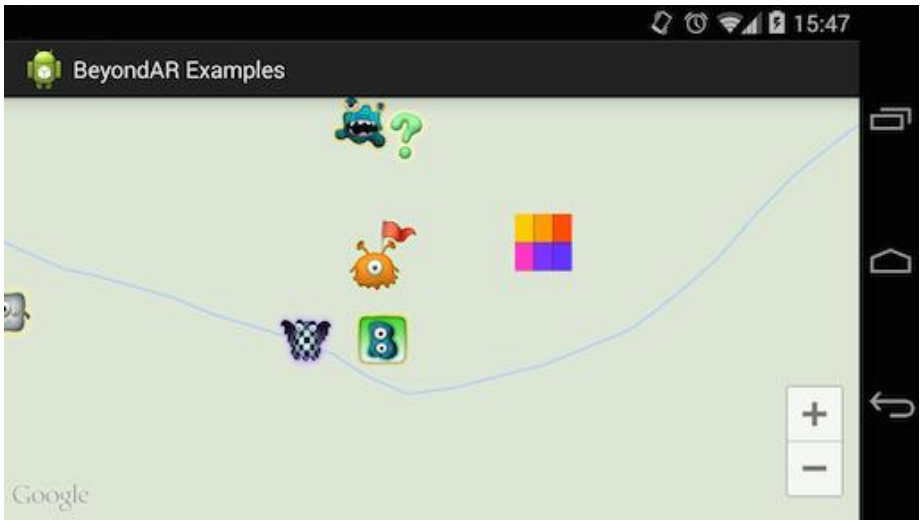


Ilustración 20: BeyondAR Plugin para Google Maps (FUENTE: github.com/BeyondAR)

En el caso de que, cuando pulsemos sobre uno de los dibujos, nos ofrezca una respuesta, lo haremos mediante un listener. En este caso, al pulsar sobre el objeto virtual aparecerá un mensaje.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
        // ...
        mMap.setOnMarkerClickListener(this);
        // ...
}

@Override
public boolean onMarkerClick(Marker marker) {
        // To get the GeoObject that owns the marker we use the following
        // method:
        GeoObject geoObject = mGoogleMapPlugin.getGeoObjectOwner(marker);
        if (geoObject != null) {
                Toast.makeText(this, "Click on a marker owned by a GeoOject
with the name: " + geoObject.getName(), Toast.LENGTH_SHORT).show();
        }
        return false;
}
```

2.1.4.1.5. Sensores

A la hora de ofrecer la experiencia AR, esta arquitectura hace el uso del acelerómetro, y del sensor del campo magnético.

El acelerómetro es capaz de percibir las aceleraciones del dispositivo, así como la posición de este, y si se sostiene de modo vertical u horizontal.

El sensor de campo magnético obtiene la fuerza del campo magnético terrestre, ayudándonos a delatar nuestra posición.

La framework nos permite notificar cualquier cambio en los sensores con un listener:

```
// Register the sensor listener
BeyondarSensorManager.registerSensorListener(mySensorListener);

// When we don't need to listen the sensors we need to unregister our
listener
BeyondarSensorManager.unregisterSensorListener(mySensorListener);
```

Cuando el dato recibido por el sensor cambia, la función onSensorChanged es llamada automáticamente. Introduciendo el objeto SensorEvent en la función, podremos averiguar si el sensor que ha cambiado es el acelerómetro o el magnético[12].

```
@Override
public void onSensorChanged(float[] filteredValues, SensorEvent event)
{
    switch (event.sensor.getType()) {
    case Sensor.TYPE_ACCELEROMETER:
        mLastAccelerometer = filteredValues;
        break;
    case Sensor.TYPE_MAGNETIC_FIELD:
        mLastMagnetometer = filteredValues;
        break;
    }
    if (mLastAccelerometer == null || mLastMagnetometer == null)
        return;

    boolean success = SensorManager.getRotationMatrix(mR, null,
mLastAccelerometer, mLastMagnetometer);
    SensorManager.getOrientation(mR, mOrientation);
    if (success)
        rotateView((float) Math.toDegrees(mOrientation[0]));
}
```

2.1.4.1.6. Radar

Tendremos la oportunidad de situar un radar en el fragmento de la cámara, que nos sea capaz de mostrar los elementos que tenemos a nuestro alrededor[13].

Para ello, deberemos posicionar gráficamente un icono con apariencia de radar en el layout de nuestra aplicación, con una estructura parecida a la siguiente:

```
<FrameLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="right|top"
    android:background="@drawable/radar_bg_small" >

    <com.beyondar.android.plugin.radar.RadarView
        android:id="@+id/radarView"
```

[12] https://github.com/BeyondAR/beyondar/wiki/Sensors
[13] https://github.com/BeyondAR/beyondar/wiki/radar-plugin

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/radar_north_small" />
</FrameLayout>
```

Este radar tendrá que ser inicializado. Creando un objeto de la clase RadarView, que está ya incluida en la framework del programa, podremos crear un plugin incluyendo ese radar, fijando la distancia del dibujo, y finalmente añadiéndolo al objeto de la clase World.

```
public void onCreate(Bundle savedInstanceState) {
    ...
    radarView = (RadarView) findViewById(R.id.radarView);
    // Create the Radar plugin
    mRadarPlugine = new RadarWorldPlugine(contex);
    // set the radar view in to our radar plugin
    mRadarPlugine.setRadarView(mRadarView);
    // Set how far (in meters) we want to display in the view
    mRadarPlugine.setMaxDistance(100);
    // and finally let's add the plugin
    mWorld.addPlugine(mRadarPlugine);
    ...
}
```

2.1.4.1.7. Location Manager

El elemento principal de nuestra aplicación, que veremos mucho más detalladamente posteriormente.

La clase BeyondarLocationManager nos ofrece todas las herramientas para obtener la posición geográfica del usuario[14].

Para usar esta clase, es necesario que la aplicación tenga el permiso de ACCESS_FINE_LOCATION y ACCESS_COARSE_LOCATION en el manifiesto.

```
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Por supuesto, el uso de esos permisos conlleva el gasto extra de batería. Es por ello por lo que en nuestra aplicación haremos que la localización se desactive cuando no se usa, de la siguiente manera:

```
@Override
void onResume(){
    // Enable GPS
    BeyondarLocationManager.enable();
}

@Override
void onPause(){
    // Disable GPS
    BeyondarLocationManager.disable();
}
```

[14] https://github.com/BeyondAR/beyondar/wiki/location-manager

La clase BeyondarLocationManager debe tener permisos de manejar los sensores de posición de nuestro terminal. Para conseguir tal cosa, debe conectar con el llamado Sistema de Servicio de Localización, que en este caso será el sistema de localización que nos ofrezca el contrato y/o operadora que tengamos contratada.

```
void onCreate(Bundle savedInstanceState){
    // ...
    BeyondarLocationManager.setLocationManager((LocationManager)
this.getSystemService(Context.LOCATION_SERVICE));
    // ...
}
```

Las propias librerías de BeyondAR vienen con un LocationManager configurado, de forma que el GPS viene incluido en la aplicación. Aun así, se tomó la decisión de crear desde cero nuestro propio sistema de localización, ya que el que venía incluido no tenía la precisión ni rapidez necesaria (3.3).

2.1.5. Android

Android es un sistema operativo basado en el kernel de Linux, diseñado principalmente para dispositivos móviles de pantalla táctil, Smartphones, tablets, y actualmente para relojes inteligentes, televisores, gafas de realidad aumentada, e incluso automóviles. (Wikipedia - Android)

Ilustración 21: Evolución de Android (FUENTE: blog.staffcreativa.pe)

Respaldada económicamente por Google y adquirida posteriormente en 2005, fue presentada al mundo junto a la fundación del Open Handset Alliance en 2007 (un consorcio de compañías de software, hardware y telecomunicaciones de estándares abiertos).

Este sistema operativo de código abierto, está desarrollado en C y C++, para el núcleo y bibliotecas de terceros, respectivamente, así como la UI en Java.

2.1.5.1. Arquitectura en Android

El sistema operativo Android está formado por la siguiente arquitectura, así como su ciclo de vida. Todo ello está recogido en el curso online de (Tushinsky).

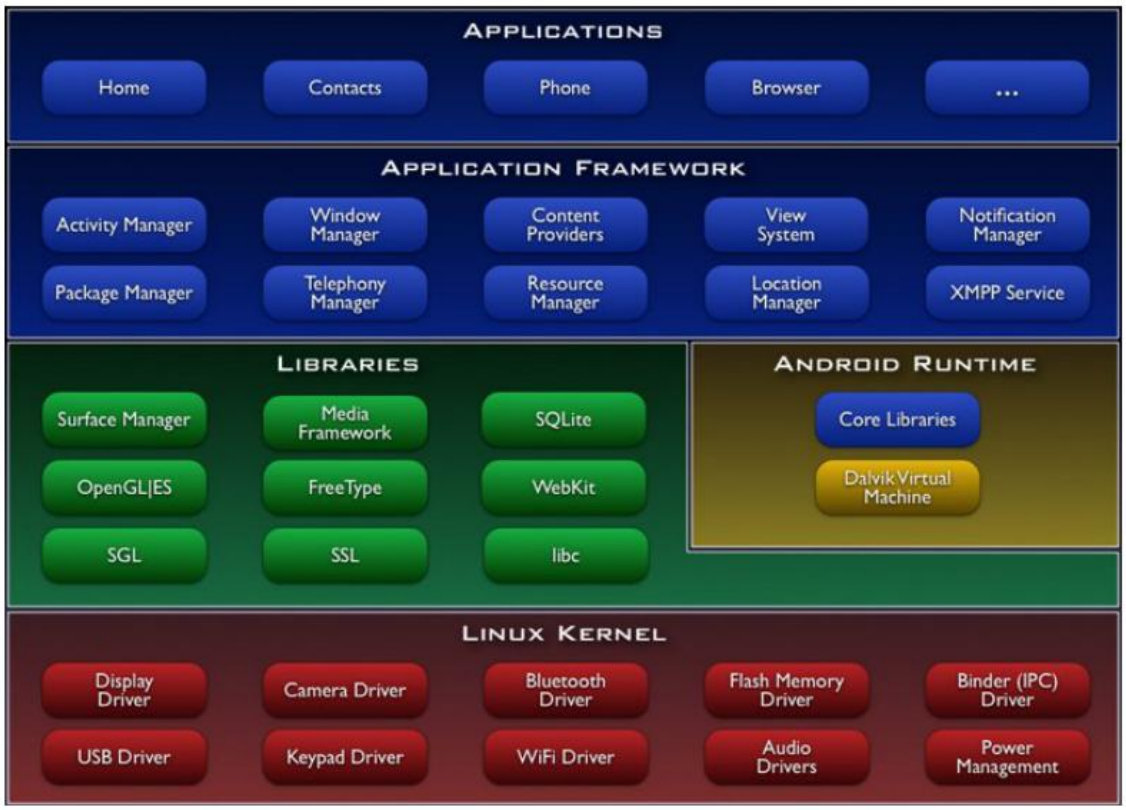


Ilustración 22: Arquitectura en el SO Android (FUENTE: androidety.com)

Con el sistema Linux como núcleo, Android está constituido por una tecnología estable y sólida, que se encarga de administrar elementos como la memoria, la red y distintas actividades de servicios y procesos.

2.1.5.1.1. Librerías Android

En el caso de las librerías, Android las usa en C/C++ para distintos componentes de la "Application Framework". A continuación describiremos los aspectos más importantes de la arquitectura.

- **System C Library**: Es una variación BSD[15] de la librería C estándar (libc), adaptada a sistema Linux,
- **Media Libraries**: Esta librería soporta la reproducción y grabación de archivos de video y audio, así como imágenes estáticas. Es un conjunto de codecs (MPEG4, H.264, MP3, AAC, AMR, JPG…)
- **Surface Manager**: Se encarga del acceso a la pantalla y de la representación de gráficos en 2D y 3D.
- **LibWebCore**: Librería de navegador de internet que otorga las herramientas necesarias para mostrar sitios web.
- **SGL**: Motor gráfico 2D.

[15] Berkeley Software Distribution.

- **3D Libraries**: Una implementación de las librerías OpenGL. Se dedican a la aceleración grafica 3D, optimizando en gran medida el renderizado grafico de más nivel.
- **FreeType**: Renderizado de fuentes tipográficas y gráficos vectoriales.
- **SQLite**: Base de datos usada en todas las aplicaciones del sistema Android

2.1.5.1.2. Application Framework

La Framework de Android nos da las funcionalidades necesarias para construir aplicaciones. Las más importantes son:

- **Activity Manager**: Controla el ciclo de vida de la aplicación.
- **Resource Manager**: Controla los recursos de la aplicación (imágenes, sonidos, archivos HTML…). En Android es llamada la clase R.
- **Location Manager**: Controla los servicios de localización.
- **Notificación Manager**: Controla los servicios de avisos, alertas y mensajes.

2.1.5.1.3. Detalles de una aplicación

Todas las aplicaciones en Android se componen de:

- **Activity**: Una actividad es lo que se muestra en la pantalla. Las aplicaciones suelen consistir de varias actividades. Cada actividad es definida en el manifiesto (AndroidManifest.xml). Para llamar a una actividad, se utilizan los Intentos, o Intents.
- **Intents**: Un intent es una llamada a una actividad, pública o privada, dentro de la aplicación. Los Intents tienen la capacidad de transmitir datos entre distintas actividades. Por ejemplo, imaginemos que en una aplicación debemos seleccionar una imagen de nuestra galería para poner una foto de perfil. Gracias a los Intents, desde una aplicación podemos abrir la galería, seleccionar una foto, y llevar dicha imagen vuelta atrás a nuestra aplicación para usarla.
- **Services**: Un service o servicio es una tarea que se ejecuta en el trasfondo de la aplicación. Existen dos tipos de servicios: Primero, los “started”, diseñados para que se ejecuten fuera, aparte de cualquier actividad, aunque sigue pudiendo interactuar con la interfaz si es necesario. Y segundo, los “bound”, diseñados para trabajar y ejecutarse en la aplicación, y estos son destruidos cuando la aplicación también es destruida.
- **Content Providers**: También conocidos como API’s, se encargan de transmitir una serie de información a través de la aplicación, de manera integrada. Por ejemplo, Google Calendar, o Google Maps (como en el actual proyecto) puede integrarse un una aplicación a través de estos proveedores de contenido.

2.1.5.1.4. Ciclo de vida de una aplicación

Toda aplicación Android se componen de una serie de métodos que definen el ciclo de vida. Cada método se ejecuta automáticamente cuando se recibe la acción (Google - Android Developer)

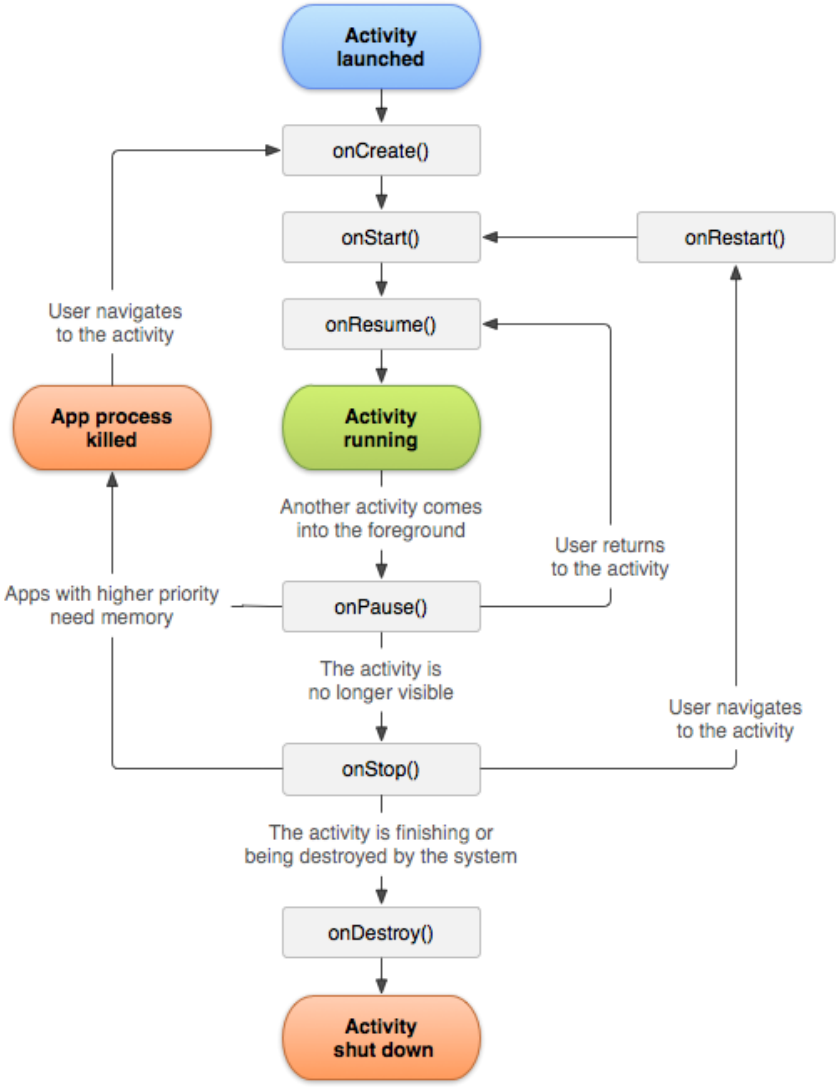


Ilustración 23: Ciclo de vida de una aplicación (FUENTE: developer.android.com)

- **onCreate**: Llamado cuando la aplicación empieza.
- **onResume**: Llamado cuando se vuelve a abrir la aplicación, ya abierta anteriormente. Utilizado para restablecer la configuración del usuario.
- **onPause**: Llamado cuando la aplicación se pausa una vez abierta. Utilizado para guardar las configuraciones de usuario.
- **onStart**: Solo es llamado cuando la aplicación se reinicia o es lanzada por primera vez.
- **onDestroy**: Cuando se llama este método, la aplicación se destruye, y solo con onCreate se puede inicializar de nuevo.

2.1.5.1.5. Sucesión de actividades en una aplicación

Las aplicaciones se organizar mediante una serie de actividades "encajonadas". En el siguiente diagrama, podemos ver como la actividad que está más arriba es la que se está ejecutando. Esa actividad está activa. El resto están en el "background" o trasfondo de la aplicación (con onPause u onStop), y corren el riesgo de que sean destruidas por el sistema en caso de que la memoria del dispositivo se vea alcanzada por el límite.

Según el ejemplo, cuando pulsamos el botón "back" de nuestro terminal, la actividad que está ejecutándose es destruida y se vuelve a la actividad anterior.

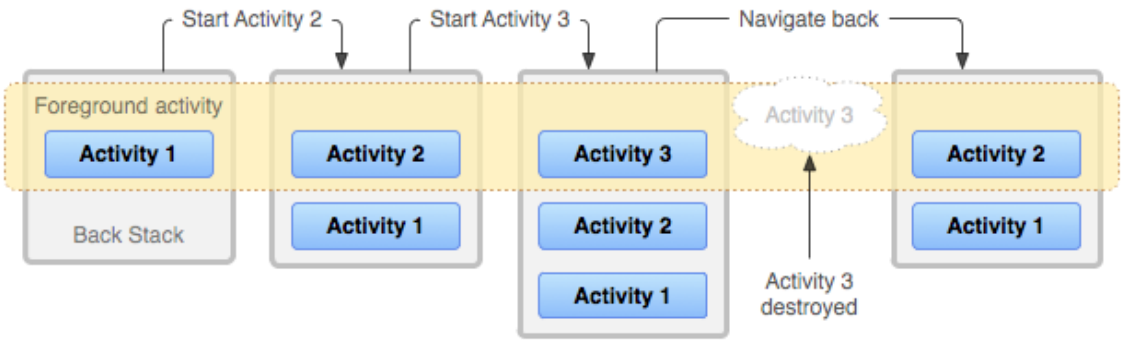


Ilustración 24: Actividades en Android (FUENTE: Curso Android, por Alex Tushinsky)

2.1.6. Servicios de Localización en Android (GPS)

A día de hoy, prácticamente todos los teléfonos modernos son capaces de rastrear la posición del terminal mediante el sistema GPS (Global Positioning System). Esta tecnología, conocida comúnmente en inglés como Location-Based Services, la usaremos para el desarrollo de la aplicación.

Ilustración 25: GPS en Android (FUENTE: gpstrackit.com)

Aunque el marco de trabajo de BeyondAR disponga de un servicio de localización propio, nosotros construiremos uno totalmente aparte para mayor precisión de este.

El sistema de GPS en Android consta de dos elementos principales:

Location Manager: Usado para obtener la posición del usuario, su movimiento, y para localizar un proveedor del servicio.

Location Providers: El proveedor que nos da la posición (puede ser el propio sistema GPS, una red WI-FI, etc.)

El sistema GPS tiene una precisión mucho mayor respecto al resto (de 5 a 15 metros). En cambio, el resto de sistemas gozan de una precisión mucho menor que puede alcanzar un error de 800 metros.

2.1.6.1. Permisos de desarrollo

Para definir un sistema GPS primero tenemos que decir de qué permisos tendremos que disfrutar en el manifiesto del programa (AndroidManifest.xml).

```
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Realmente esto sería una redundancia, ya que el permiso de ACCESS_FINE_LOCATION ya incluye el permiso de ACCESS_COARSE_LOCATION como es lógico, pero no al revés.

Si además de eso, tenemos planeado usar un sistema GPS apoyado con Google Maps necesitaremos permisos para acceder a Internet.

```
<uses-permission android:name="android.permission.INTERNET" />
```

2.1.6.2. Actividad principal

Gracias a la interfaz LocationListener, nos añadirá a la clase una serie de métodos requeridos para el sistema de localización.

```
public class LBSActivity extends Activity implements LocationListener
{
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
}
public void onLocationChanged(Location location) {
}
public void onProviderDisabled(String provider) {
}
public void onProviderEnabled(String provider) {
}
public void onStatusChanged(String provider, int status, Bundle
extras) {
}
```

- **onLocationChanged**: Este método es llamado cuando una nueva latitud y longitud es recibida por la terminal. Estas actualizaciones de posición llegaran en función del movimiento del teléfono.
- **onProviderEnabled**: Se trata de encender o iniciar el acceso de localización con el proveedor.
- **onProviderDisable**: La acción contraria a la anterior. Podremos finalizar nuestra conexión con el proveedor llamando a este método.
- **onStatusChanged**: Este método se llamada cuando el estado del proveedor cambia. Por ejemplo, si mientras caminamos (siempre con el proveedor activado), la señal GPS se pierde temporalmente.

2.1.6.3. Configuración del sistema GPS

Para empezar a trabajar con un sistema LBS, lo primero es crear la instancia de nuestra clase LocationManager:

```
LocationManager oLBSManager = (LocationManager)
getSystemService(LOCATION_SERVICE);
```

Posteriormente, configuraremos la clase Criteria, que definirá como será nuestro GPS en muchos aspectos (precisión, uso de energía…), y una variable tipo String que nos dará el proveedor que más se asemeja a ese criterio.

```
Criteria oCriteria = new Criteria();
oCriteria.setAccuracy(Criteria.ACCURACY_FINE);
oCriteria.setCostAllowed(false);
oCriteria.setPowerRequirement(Criteria.POWER_HIGH);
oCriteria.setSpeedRequired(false);
oCriteria.setAltitudeRequired(true);
oCriteria.setBearingRequired(false);
String sProvider = oLBSManager.getBestProvider(oCriteria, true);
```

Para acabar, solo nos queda probar este proveedor. Para ello, podemos chequear las actualizaciones de localización. En el siguiente código, si el proveedor existe, nos mandará una actualización cada 1000 milisegundos, o cada 1 metro. Dependiendo de esta configuración la precisión y el gasto de batería variaran significativamente.

```
if (sProvider != null) {
oLBSManager.requestLocationUpdates (sProvider, 1000, 1, this);
// or use LocationManager.GPS_PROVIDER instead of sProvider variable
} else {
Toast.makeText(getBaseContext(), "No GPS signal",
Toast.LENGTH_SHORT).show();
finish();
}
```

2.1.7. Localización con Google Maps

La funcionalidad que nos ofrece Google Maps es la mejor en su área. Potente y fácil, nos permite crear sistemas de localización sobre un mapa.

El proceso para usar un mapa de Google se hará mediante la obtención de una API Key. Para ello, los pasos son los siguientes:

Primero debemos instalar los Google Play Services en la librería de Eclipse, mediante el SDK Manager.

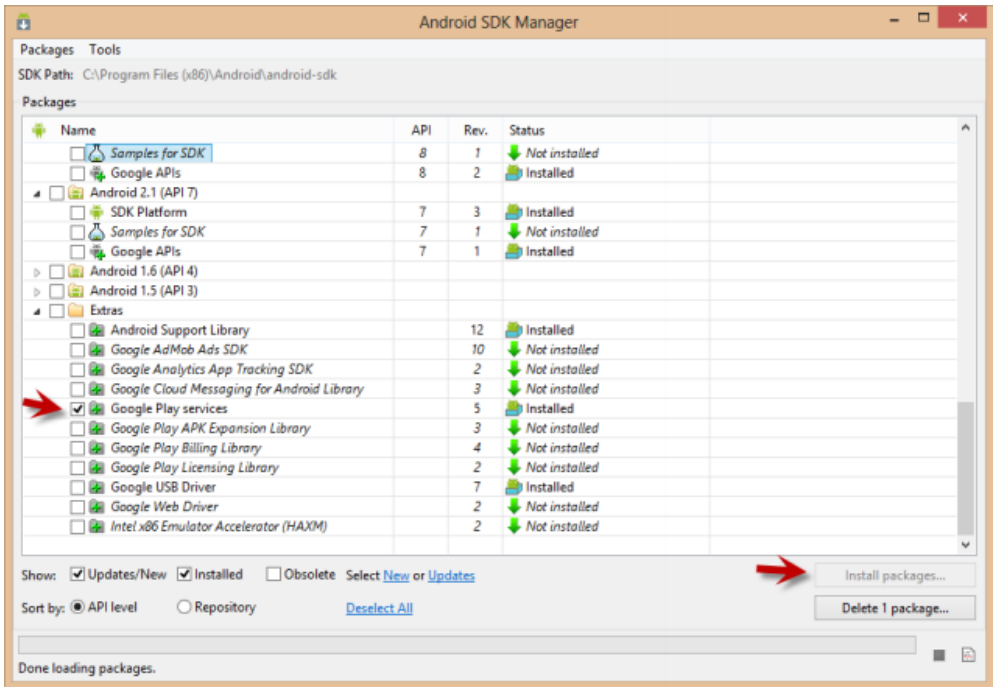


Ilustración 26: Instalación Google Play Services (FUENTE: Eclipse)

Ahora deberemos obtener la API Key. Para ello debemos conocer la llamada SHA-1 fingerprint, un identificador de nuestro entorno de trabajo. Esto viene dado por los certificados de la aplicación; Cuando está en desarrollo, tenemos un certificado, pero si un día lanzamos la aplicación al mercado, la aplicación será compilada con otro certificado o "keystore". Esta llave creara su propio SHA-1 fingerprint.

Para obtener la huella o fingerprint, necesitamos la keystore (llamada debug.keystore) y la keytool de Java.

Para obtener la primera:

- Windows 7 / 8 - C:\Users\<user>\.android\debug.keystore
- Windows XP - C:\Documents and Settings\<user>\.android\debug.keystore

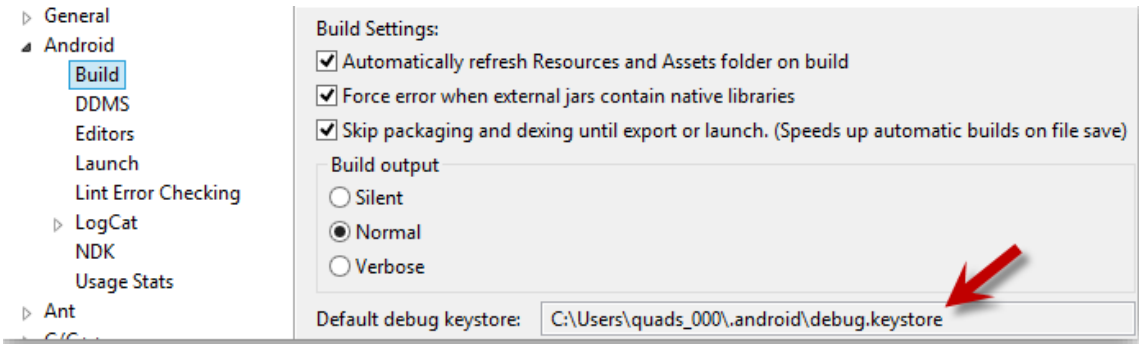


Ilustración 27: Ubicación debug.keystore (FUENTE: Eclipse)

La segunda se encontrara en C:\Program Files(x86)\Java\JDKxxx\bin\.

Una vez tenemos ambos archivos localizados, deberemos crear un archivo batch con los siguientes comandos.

"Ruta de la keytool de Java aquí" -list -v -keystore

"Ruta de la debug.keystore aqui" -alias androiddebugkey -storepass android -keypass android pause

Nuestro archivo Batch será como el siguiente, y en el podremos obtener la huella, con el nombre de SHA1.

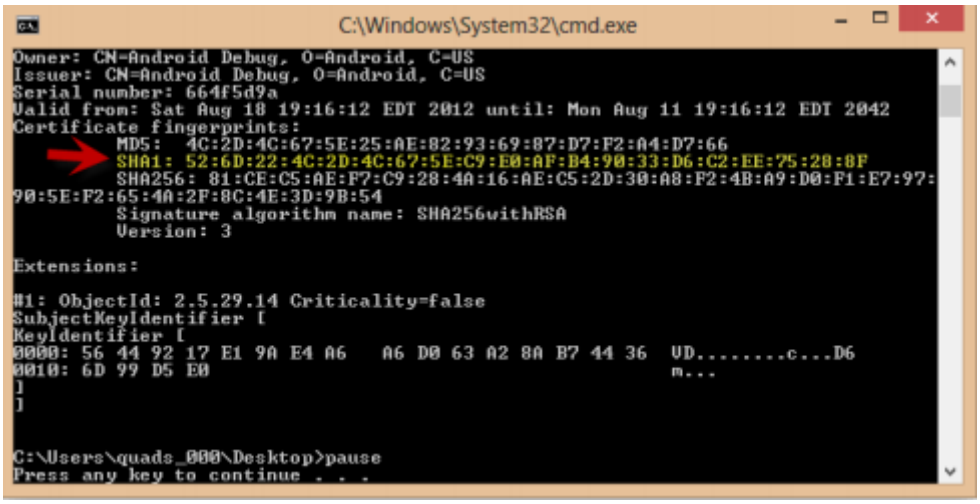


Ilustración 28: Archivo .batch y SHA1 (FUENTE: Eclipse)

Ahora ha llegado el momento de obtener la API Key para usar los mapas de Google.

Desde la consola de desarrollo de Google, crearemos un nuevo proyecto. En este caso el nuestro se llamará PFG-Google Maps API v2.

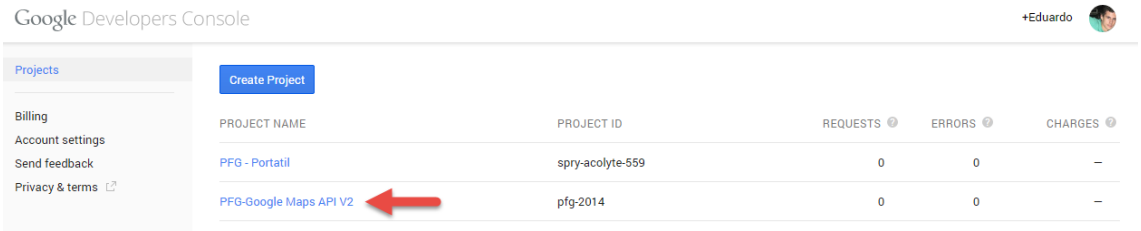


Ilustración 29: Proyecto Google Developers Console (FUENTE: Google)

Dentro de este proyecto y de la pestaña APIs & auth, en la opción APIs, activaremos Google Maps Android API v2.

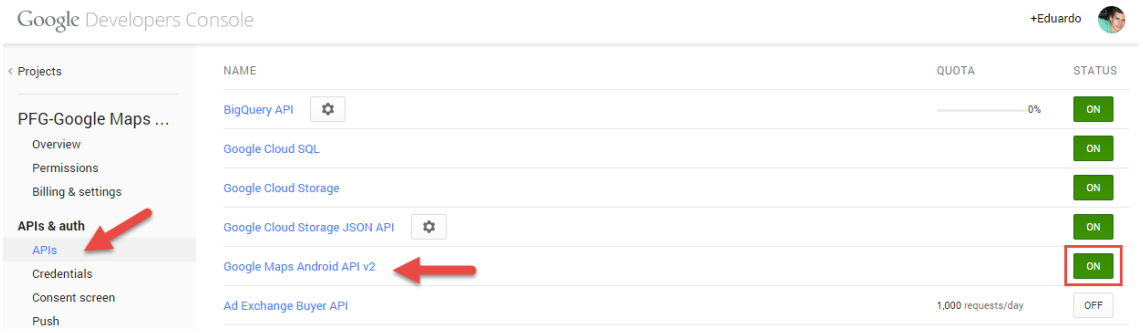


Ilustración 30: Activación API Google Maps v2 (FUENTE: Google)

En la misma pestaña, opción Credentials, veremos un párrafo que pone Public API Access. Dentro, hacemos click en la opción Create New Key. Luego, para este caso, seleccionamos la sub-opción Android Key.

Nos aparecerá un campo de texto. Si leemos las instrucciones, está claro: En él, escribiremos (o mejor dicho, copiaremos) la SHA-1 fingerprint, y luego de manera seguida, punto y coma y el nombre del paquete de nuestra aplicación.

Si le damos a añadir, se creara una API Key que podremos copiar a nuestro manifiesto Android para permitir el uso del mapa.

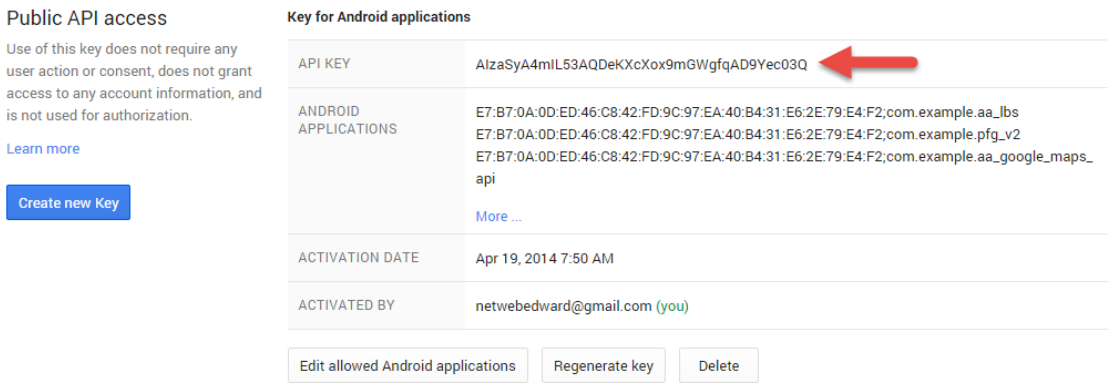


Ilustración 31: APY Key (FUENTE: Google)

Una vez tenemos la Key, es hora de volver a Eclipse.

Para que nuestra aplicación pueda usar los servicios de Google, necesitamos implementar la librería que descargamos al principio en el proyecto.

La librería funcionará como un proyecto aparte. Para hacerlo, abriremos un nuevo proyecto Android, usando la opción Android Proyect from Existing Core. Navegaremos hasta encontrar la librería que estará en: /extras/google/google-play-services_lib.

Este proyecto-librería se usara en todas las aplicaciones que utilicen Google Maps API.

Cuando tengamos una aplicación y queramos añadir esta librería, haremos click derecho en el proyecto desde el explorador de archivos situado a la izquierda del programa, y desde la

opción Properties, en la pestaña Android, veremos que podemos añadir librerías en la sección Library. Hacemos click en Add y seleccionamos el proyecto que acabamos de importar.

Si lo hacemos bien, saldrá un símbolo de check verde a la izquierda de la ruta.

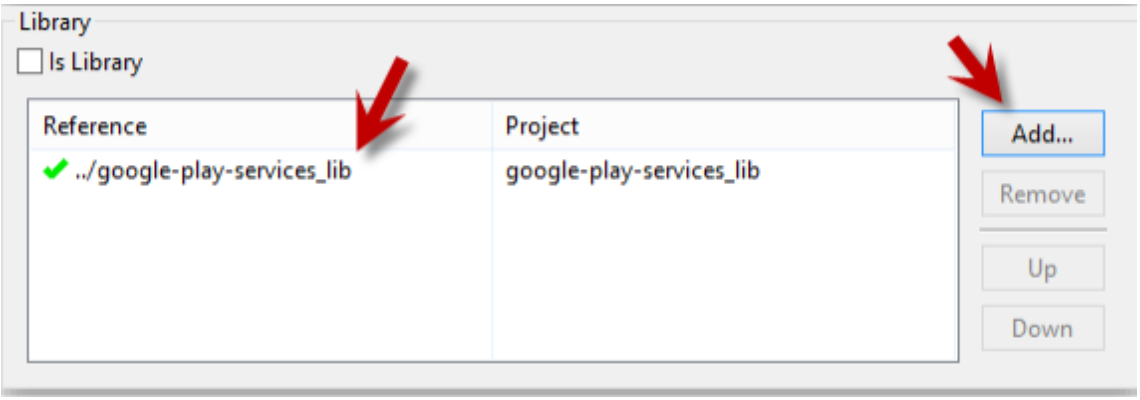


Ilustración 32: Incluir librería Google Play (FUENTE: Eclipse)

Para acabar, modificaremos nuestro AndroidManifest.xml. Dentro del segmento <application> añadiremos lo siguiente:

```
<meta-data android:name="com.google.android.maps.v2.API_KEY"
android:value="Insertar clave API aquí " />
```

Donde el valor de la API será el código que obtuvimos anteriormente.

Para acabar, Google nos ofrece dos herramientas de exploración geográfica que nos serán de gran utilidad: Maps y Earth.

Google Maps nos muestra imágenes de mapas desplazables, fotografías por satélite y lo más importante para nosotros, rutas entre diferentes ubicaciones.

Google Earth es una variante a nivel entorno de escritorio que nos da las mismas e incluso más posibilidades que Google Maps a nivel de exploración, desplazamiento, medición geográfica, etc.

Nosotros hemos utilizado estas aplicaciones para crear la ruta de seguimiento, como veremos más adelante.

2.2. Requisitos

Para el desarrollo de esta aplicación, los requisitos establecidos se basan tanto en el sistema operativo en el que se ejecuta el programa, como en la funcionalidad de esta.

Como programa, la API mínima que soporta nuestra aplicación es la nivel API 8, es decir, Android 2.2. Como versión recomendada, y por tanto la que utilizaremos será la API 17, es decir Android 4.2.

Acerca de las funcionalidades, la aplicación debe informar de la posición de distintos elementos en el campus de la Escuela Politécnica de la UC3M, como pueden ser la Biblioteca, el Salón de Grados, y otros edificios de importancia.

2.3. Restricciones marco regulador

A continuación detallaremos una serie de restricciones que nos limitaran y guiaran durante el desarrollo de la aplicación.

Como primera restricción, se prestó atención en el uso del SDK de Realidad Aumentada. Era necesario que fuera Open Source[16], y que no implicara el pago de licencias de ningún tipo, además de ser totalmente libre y abierta al público. BeyondAR cumplía a la perfección todos estos requisitos, por esa razón fue la elegida.

La segunda restricción/es vendría dará en el caso de que la aplicación fuera subida al Google Play Market[17]. Una aplicación subida a dicha tienda debe cumplir una serie de restricciones y acuerdos de distribución para desarrolladores[18].

En nuestro caso no subiremos la aplicación al Google Play Market, pero cumpliremos las distintas restricciones.

[16] Open Source Initiative
[17] Google Play Market
[18] Acuerdo de distribución para desarrolladores

3. Diseño de la solución técnica

A continuación, describiremos los procesos empleados para el desarrollo íntegro del proyecto. Para la creación del programa, llevaremos a cabo diferentes fases, que se estipularán de manera cronológica.

3.1. Construcción del entorno

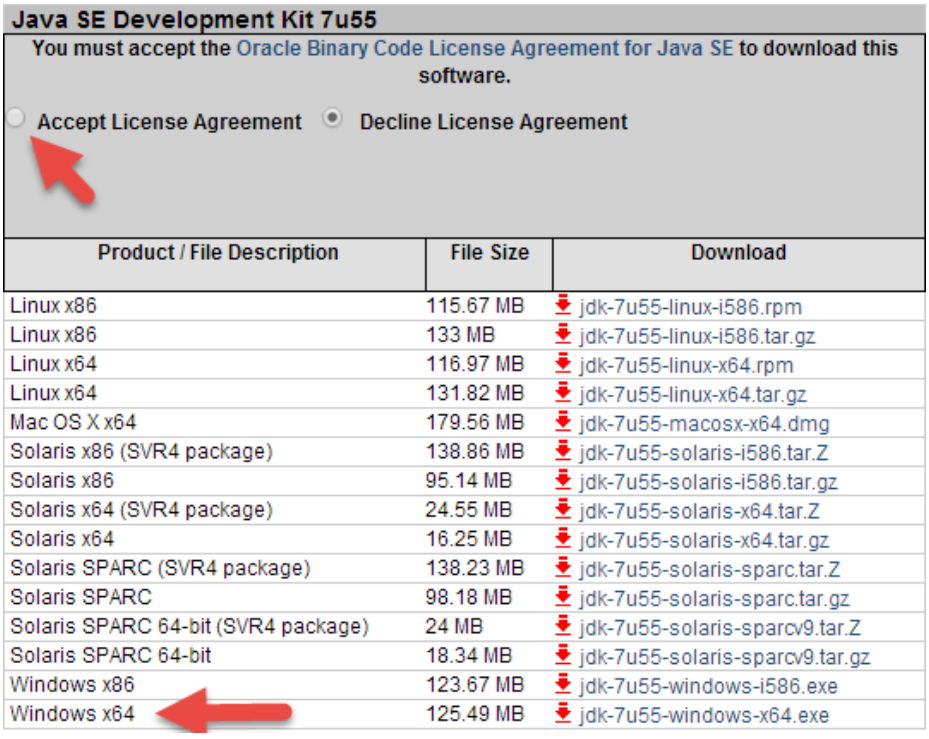
Para la realización del proyecto, se tuvo que preparar el equipo con las herramientas de trabajo necesarias.

Google ha hecho que la instalación del entorno sea lo más sencilla posible, poniendo a nuestra mano una configuración unificada si así lo queremos. Existen dos maneras de realizar el proceso, una pre-configurada, y otra manual.

- Paso 1: Descargas

Empezaremos descargando el software requerido.

1. Java JDK está disponible en el siguiente enlace: http://www.oracle.com/technetwork/java/javase/downloads/index.html



Java SE Development Kit 7u55

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux x86	115.67 MB	jdk-7u55-linux-i586.rpm
Linux x86	133 MB	jdk-7u55-linux-i586.tar.gz
Linux x64	116.97 MB	jdk-7u55-linux-x64.rpm
Linux x64	131.82 MB	jdk-7u55-linux-x64.tar.gz
Mac OS X x64	179.56 MB	jdk-7u55-macosx-x64.dmg
Solaris x86 (SVR4 package)	138.86 MB	jdk-7u55-solaris-i586.tar.Z
Solaris x86	95.14 MB	jdk-7u55-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.55 MB	jdk-7u55-solaris-x64.tar.Z
Solaris x64	16.25 MB	jdk-7u55-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	138.23 MB	jdk-7u55-solaris-sparc.tar.Z
Solaris SPARC	98.18 MB	jdk-7u55-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	24 MB	jdk-7u55-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.34 MB	jdk-7u55-solaris-sparcv9.tar.gz
Windows x86	123.67 MB	jdk-7u55-windows-i586.exe
Windows x64	125.49 MB	jdk-7u55-windows-x64.exe

Ilustración 33: Descarga paquete JDK (FUENTE: Java.com)

Aceptaremos el acuerdo de licencia y descargamos la versión correcta para nuestra plataforma. En nuestro caso, instalamos el JDK (Java Development Kit) en su versión más demandada para Windows x64. En nuestro caso, 1.7.

2. Ahora descargaremos el SDK (Software Development Kit): http://developer.android.com/sdk/index.html

A partir de aquí, tendremos dos opciones. Si quisiéramos una solución "todo en uno", podemos descargar el ADT Bundle (Android Development Tools) desde la página de Android. Este archivo nos otorga todo lo que necesitamos a la hora de desarrollar, incluyendo Eclipse 3.8 como IDE (Integrate Development Enviroment), y la última versión del SDK de Android, todo pre-configurado para empezar a funcionar.

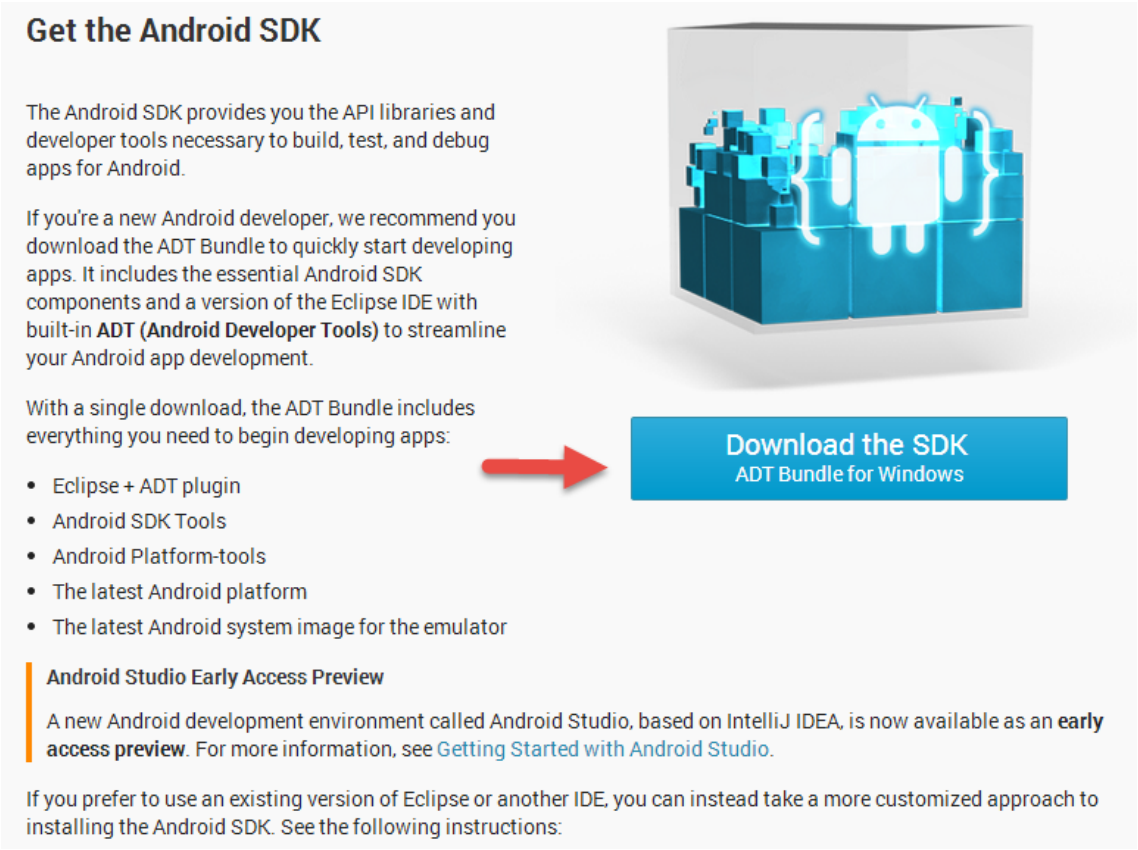


Ilustración 34: Descarga SDK Android (FUENTE: Google)

Si lo que queremos es una instalación individual de cada componente (como hemos hecho en el proyecto), procederemos de la siguiente manera. Así podremos ver en detalle cada instalación.

En este caso solo descargaremos el SDK mediante la siguiente opción en la misma página que antes.

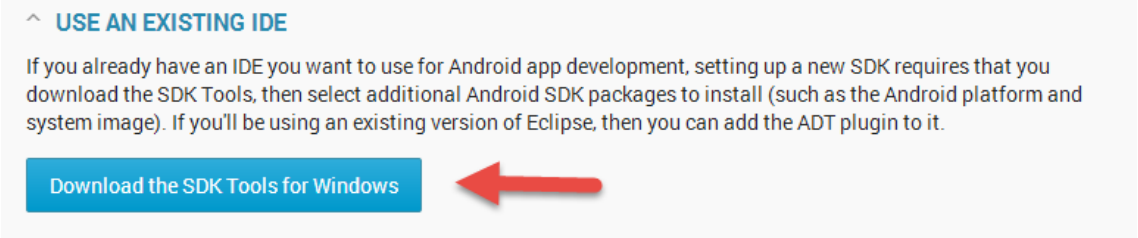


Ilustración 35: Descarga SDK Tools (FUENTE: Google)

3. Solo en el caso de que vayamos a instalar los componentes manualmente: Ahora descargamos Eclipse desde el siguiente enlace: http://www.eclipse.org/downloads/

En nuestro caso, usaremos la versión IDE for Java Developers de 64 bits, que incluirá todo lo necesario para desarrollar en Java en nuestro equipo.

Ilustración 36: Descarga IDE Eclipse (FUENTE: eclipse.com)

- Paso 2: Instalación

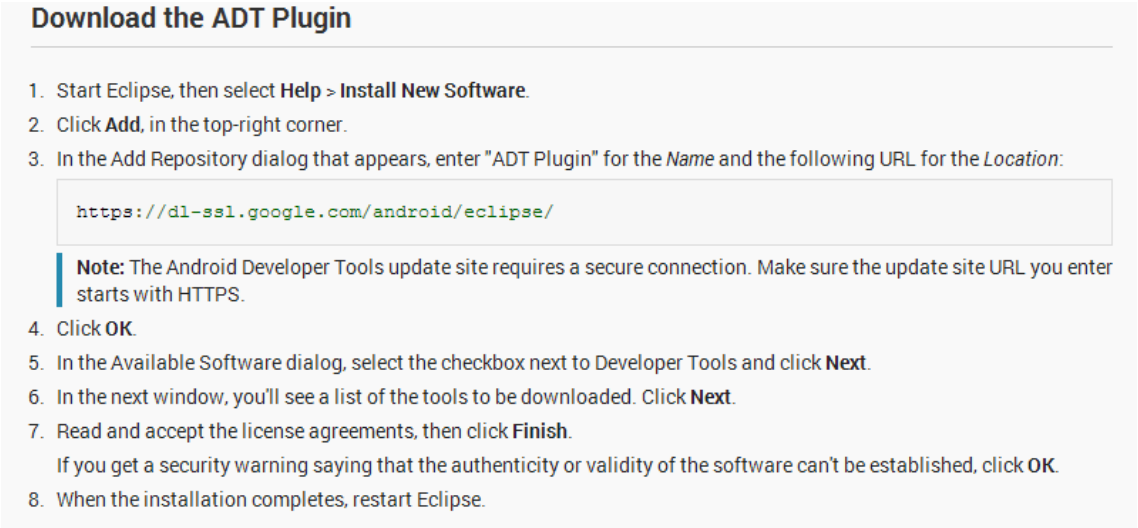
Ahora que tenemos las descargas necesarias, tenemos que configurar estos paquetes. Para ello seguiremos los siguientes pasos.

1. Instalamos el Java JDK siguiendo los pasos de la instalación Wizard. No existe ninguna opción especial que deba seleccionarse. Simplemente pulsamos "Next" hasta que lo completemos.

2. Si usamos la forma pre-configurada, nos bastará con extraer el contenido el ADT Bundle, abrir la carpeta Eclipse e iniciar el archivo .exe.

   En el modo manual, como es el caso: Primero extraemos el Eclipse IDE en la ruta que queramos (no tiene instalador), y posteriormente, instalamos el Android SDK, en este orden. Todos estos paquetes los descargamos previamente.

3. Para acabar la base de la instalación, ahora será el turno del Google ADT Plug-in, para integrar el Android SDK con Eclipse. Este plugin ADT (Android Development Tools) proveerá de un puente entre el código basado en las herramientas Android y el entorno Eclipse. Para ello, seguimos los pasos descritos en la imagen.



Download the ADT Plugin

1. Start Eclipse, then select **Help** > **Install New Software**.
2. Click **Add**, in the top-right corner.
3. In the Add Repository dialog that appears, enter "ADT Plugin" for the *Name* and the following URL for the *Location*:

   ```
   https://dl-ssl.google.com/android/eclipse/
   ```

   Note: The Android Developer Tools update site requires a secure connection. Make sure the update site URL you enter starts with HTTPS.
4. Click **OK**.
5. In the Available Software dialog, select the checkbox next to Developer Tools and click **Next**.
6. In the next window, you'll see a list of the tools to be downloaded. Click **Next**.
7. Read and accept the license agreements, then click **Finish**.
   If you get a security warning saying that the authenticity or validity of the software can't be established, click **OK**.
8. When the installation completes, restart Eclipse.

Ilustración 37: Instalación ADT Plugin (FUENTE: Google)

- Paso 3: Actualizar.

Una vez tengamos todo funcionando, dentro de Eclipse tendremos que hacer una serie de tareas:

Ilustración 38: SDK y AVD Manager's (FUENTE: Eclipse)

Android SDK Manager: Una vez tengamos todo lo necesario, deberemos configurar una serie de paquetes para el desarrollo en PC. Google ha intentado facilitar al máximo la configuración de cara a todas las versiones y librerías disponibles para Android. De esta manera, el SDK Manager nos hará las cosas mucho más sencillas.

Todos los rasgos y características de nuestro programa vendrán dados según los paquetes que hayamos descargado. Si queremos desarrollar una aplicación que soporte versión Android 4.2.2 (API 17), deberemos descargar dicho paquete. Si queremos que soporte la API de Google para incluir mapas, deberemos descargar el paquete de Google Play Services, y así sucesivamente.

Para nosotros, los paquetes que instalaremos serán los siguientes (aparte de los que vienen por defecto). Algunos de ellos los instalaremos por carácter preventivo.

Tools			
Android SDK Tools		22.6.2	Update available: rev. 22.6.3
Android SDK Platform-tools		19.0.1	Installed
Android SDK Build-tools		19.0.3	Installed
Android SDK Build-tools		*19.0.2*	*Not installed*
Android SDK Build-tools		*19.0.1*	*Not installed*
Android SDK Build-tools		*19*	*Not installed*
Android SDK Build-tools		*18.1.1*	*Not installed*
Android SDK Build-tools		*18.1*	*Not installed*
Android SDK Build-tools		*18.0.1*	*Not installed*
Android SDK Build-tools		*17*	*Not installed*
Android 4.4.2 (API 19)			
Documentation for Android SDK	19	2	Installed
SDK Platform	19	3	Installed
Samples for SDK	19	5	Installed
Android Wear ARM EABI v7a System Image	*19*	*1*	*Not installed*
ARM EABI v7a System Image	19	2	Installed
Intel x86 Atom System Image	19	2	Installed
Google APIs (x86 System Image)	19	4	Installed
Google APIs (ARM System Image)	19	4	Installed
Glass Development Kit Preview	*19*	*4*	*Not installed*
Sources for Android SDK	19	2	Installed
Android 4.3 (API 18)			
Android 4.2.2 (API 17)			
SDK Platform	17	2	Installed
Samples for SDK	17	1	Installed
ARM EABI v7a System Image	17	2	Installed
Intel x86 Atom System Image	*17*	*1*	*Not installed*
MIPS System Image	*17*	*1*	*Not installed*
Google APIs	17	3	Installed
Sources for Android SDK	17	1	Installed
Android 4.1.2 (API 16)			
Android 4.0.3 (API 15)			
Android 4.0 (API 14)			
Android 3.2 (API 13)			
Android 3.1 (API 12)			
Android 3.0 (API 11)			
Android 2.3.3 (API 10)			
Android 2.2 (API 8)			
SDK Platform	8	3	Installed
Samples for SDK	8	1	Installed
Google APIs	8	2	Installed
Android 2.1 (API 7)			
Android 1.6 (API 4)			
Android 1.5 (API 3)			
Extras			
Android Support Repository		*5*	*Not installed*
Android Support Library		19.1	Installed
Google Analytics App Tracking SDK		*3*	*Not installed*
Google Play services for Froyo		12	Installed
Google Play services		16	Installed
Google Repository		7	Installed
Google Play APK Expansion Library		*3*	*Not installed*
Google Play Billing Library		*5*	*Not installed*
Google Play Licensing Library		*2*	*Not installed*
Google USB Driver		9	Installed
Google Web Driver		*2*	*Not installed*
Intel x86 Emulator Accelerator (HAXM installer)		*4*	*Not installed*

Ilustración 39: SDK Manager (FUENTE: Eclipse)

Android Virtual Device (AVD): Para emular nuestras aplicaciones en el entorno de trabajo, haremos uso de un simulador de dispositivos. Este nos permitirá simular una grandísima variedad de tipos de hardware y versiones de Android. En nuestro caso, a lo largo del desarrollo se usaron principalmente los siguientes tipos.

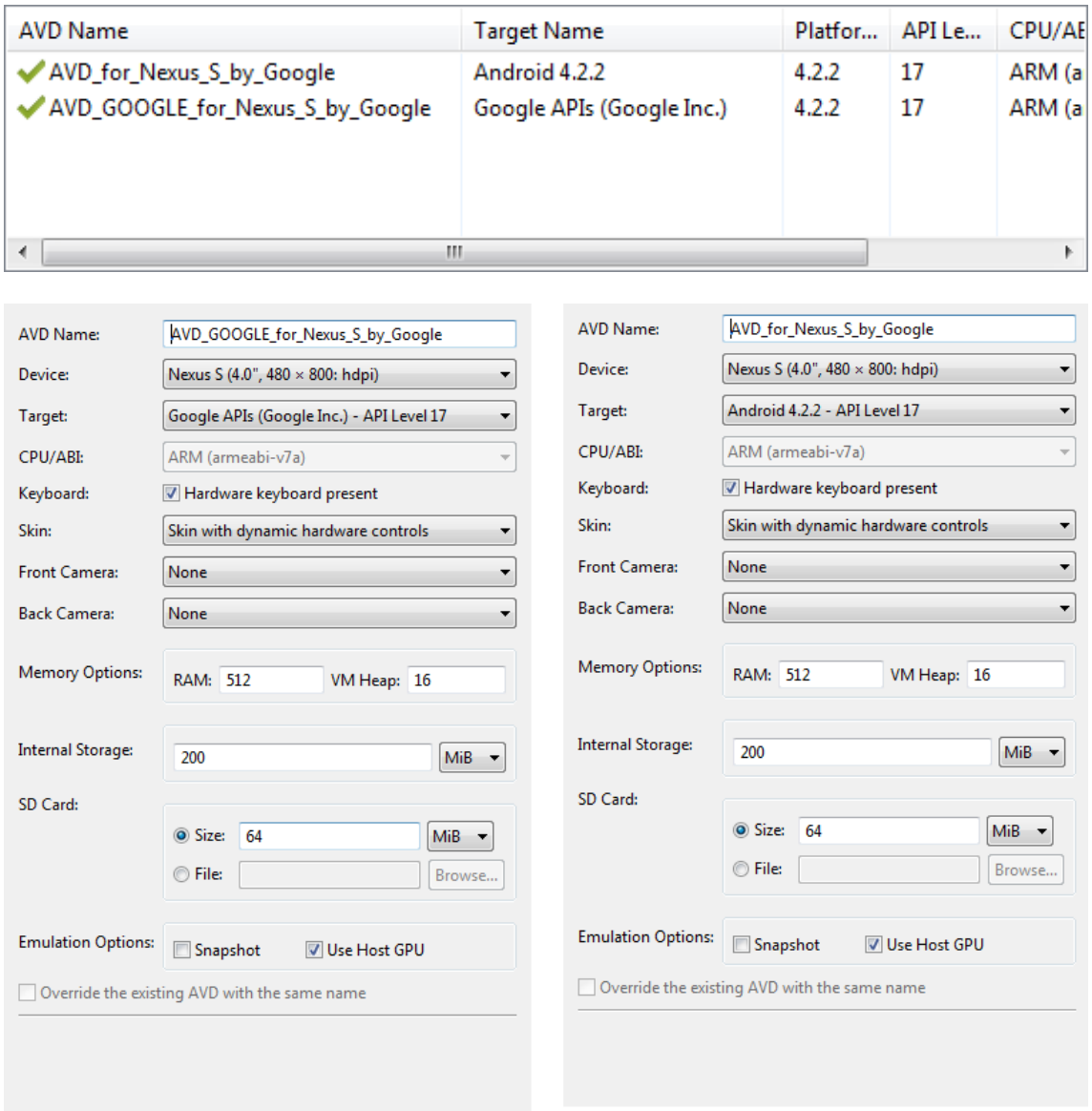


Ilustración 40: AVD Manager (FUENTE: Eclipse)

Podemos modificar los siguientes parámetros:

- **AVD Name**: El nombre para el emulador.
- **Device**: El dispositivo físico. Con el propio programa nos vienen ya una serie de plantillas de ciertos móviles que podemos usar. Pero podemos también crear las nuestras propias para crear otros dispositivos.
- **Target**: La versión del Android SDK y la versión del OS Android que usaremos en el AVD. Estas vendrán si descargamos las respectivas versiones en el SDK Manager, como explicamos anteriormente.

- **Keyboard/Skin/Cameras**: Parámetros para simular la terminal. La cámara se puede emular con una webcam.
- **Memory Options**: Cantidad de RAM y Virtual Heap Memory. Usaremos las que vienen por defecto. Hay que tener en cuenta que si ponemos más cantidad de RAM, este necesitara más rendimiento y puede ralentizar nuestra máquina. En nuestro caso 200. No confundir con memoria RAM; si ponemos más de la debida no perjudicara al ordenador.
- **Internal Storage / SD Card**: Podemos simular una memoria interna y la de una tarjeta SD. Es recomendable crear una memoria suficiente para que no haya problemas.
- **Emulator Options**: La función Snapshot nos permite aumentar la velocidad ahorrando memoria RAM una vez ha sido iniciado, y restaurándola para futuros usos, algo así como una reserva. De esta manera, cada vez que empezamos a correr el emulador, no tiene que proceder con el "boot" (que es lento).

La función Use Host GPU sirve para aplicaciones graficas pesadas, ya que el emulador se ayuda del propio ordenador.

Puedes observar que la única diferencia entre ambos emuladores es el Target. Esto es la versión del SDK y del OS que queremos que use. En el caso de que queramos incluir una API de Google, deberemos seleccionar el AVD de la izquierda.

Para terminar, ir a la pestaña Help del Eclipse, y después "Check for Updates" para ver si hay alguna versión actualizada de Eclipse disponible.

3.2. Desarrollo de una ruta y Simulación

Durante el desarrollo de la aplicación, se consideró esencial desarrollar una ruta geográfica, que nos simplificara la tarea a la hora de analizar y buscar errores en nuestra aplicación.

Sin una ruta, cada vez que intentáramos probar la aplicación deberíamos irnos al lugar donde tenemos nuestros puntos virtuales y andar alrededor; En cambio, con estas herramientas, primero crearemos una ruta en un formato que sea simulable, y posteriormente, con un simulador de rutas, "engañaremos" a la terminal para que crea que estamos caminando por dicha ruta con el GPS activado.

3.2.1. Creación de la ruta con Google Maps

Para empezar, crearemos una ruta con las herramientas provistas por Google Maps. El servicio nos ofrece la opción de "Mis sitios" que nos permitirá crear mapas con capas personalizadas: Estas capas se pueden componen de rutas (de A a B), puntos de interés, etc.

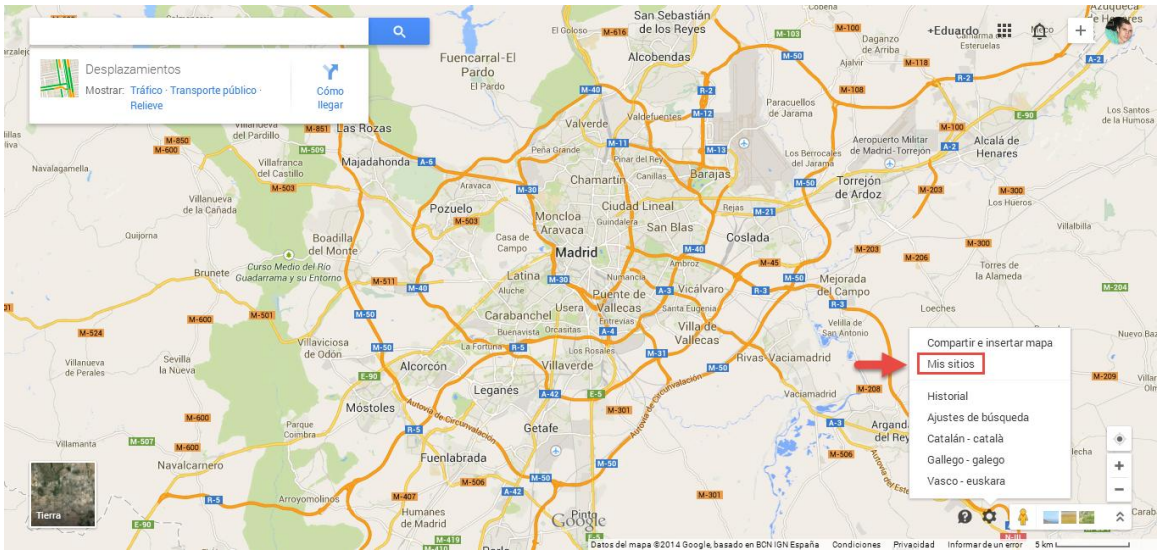


Ilustración 41: Creación ruta Google Maps (FUENTE: Google)

Pulsando sobre “Mis sitios”, se nos llevará a la interfaz donde podremos personalizar nuestras capas. Pulsando sobre Mapas podremos ver nuestros mapas guardados. En nuestro caso, Ruta1.0 será el elegido. Esta es la versión definitiva de la ruta, pero mostraremos su creación desde cero.

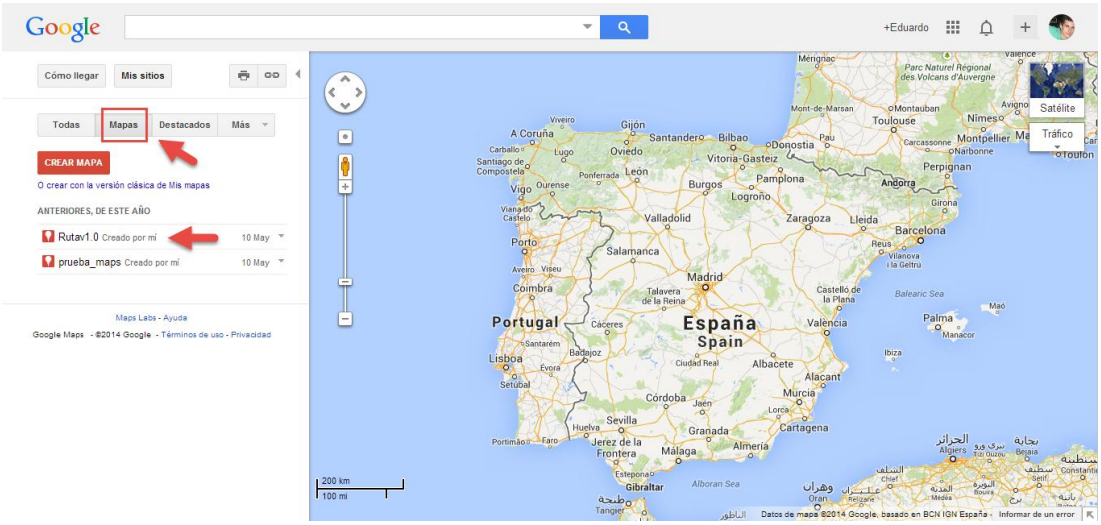


Ilustración 42: Creación mapa (FUENTE: Google)

Este será nuestro lugar de trabajo. Crearemos una ruta que pase por los principales puntos de interés del campus de la Politécnica, como las diferentes caras del edificio Sabatini, la fuente, la biblioteca, etc.

Ilustración 43: Creación mapa (FUENTE: Google)

Nuestro primer paso será añadir dichos puntos. Con la opción “Añadir marcador” iremos sumando sobre nuestra capa los puntos que queremos ver (listados en la imagen).

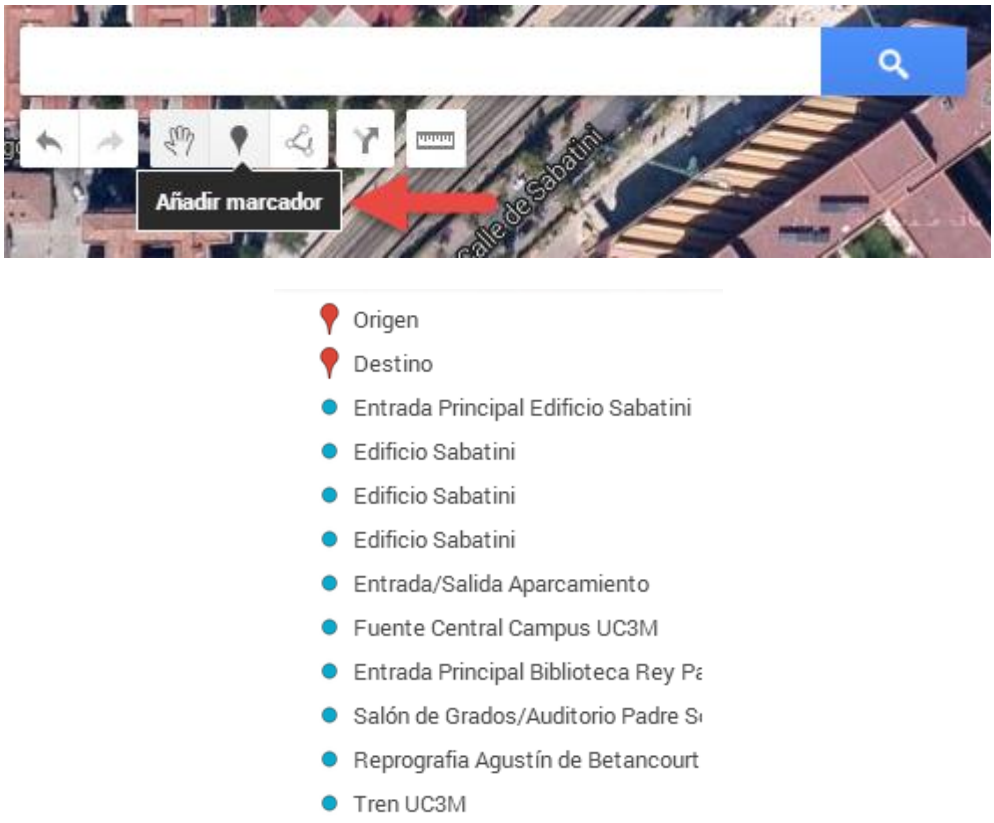


Ilustración 44: Creación mapa (FUENTE: Google)

Siendo este el resultado:

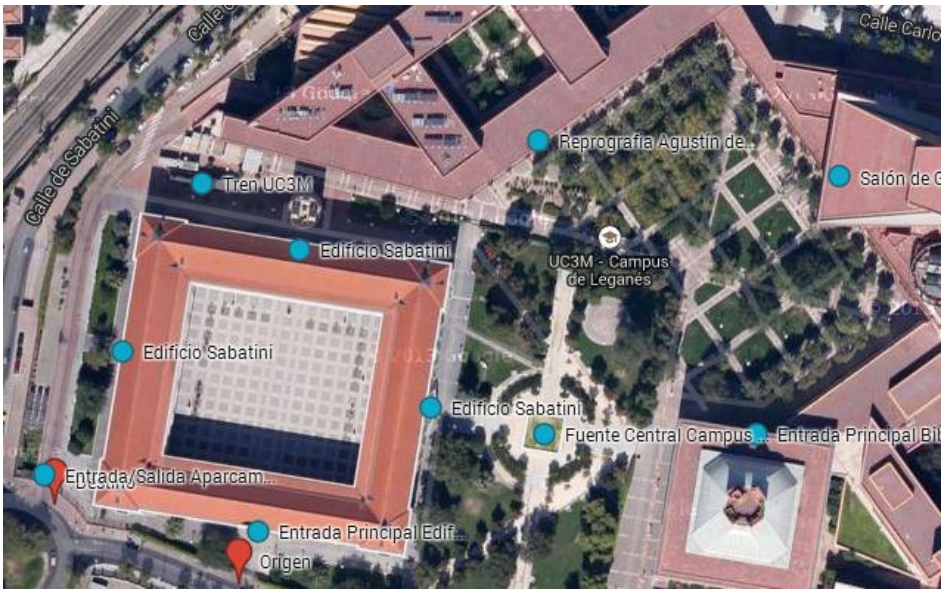


Ilustración 45: Creación mapa (FUENTE: Google)

Nuestro próximo paso, será crear la ruta. Para ello, hemos puesto dos puntos de interés llamados Origen y Destino, que nos facilitaran la tarea. Definiremos una ruta con el inicio y final en esos dos puntos.

Pulsamos sobre la opcion “Añadir indicaciones” y crearemos una capa con dos puntos: A y B.

Ilustración 46: Creación mapa (FUENTE: Google)

Nuestro punto A será el punto de Origen, y el punto B, naturalmente, el punto de Destino.

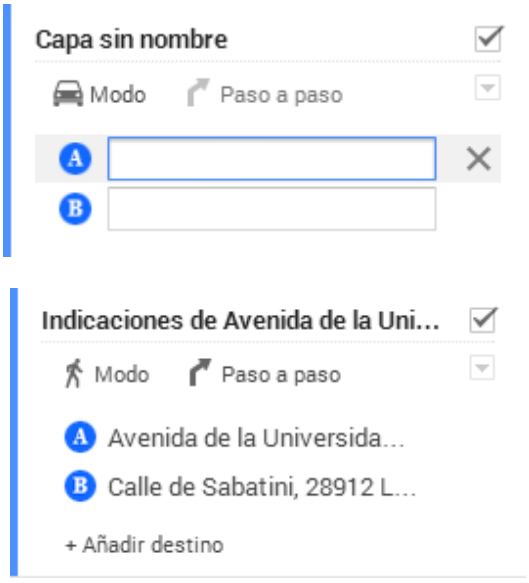


Ilustración 47: Creación ruta (FUENTE: Google)

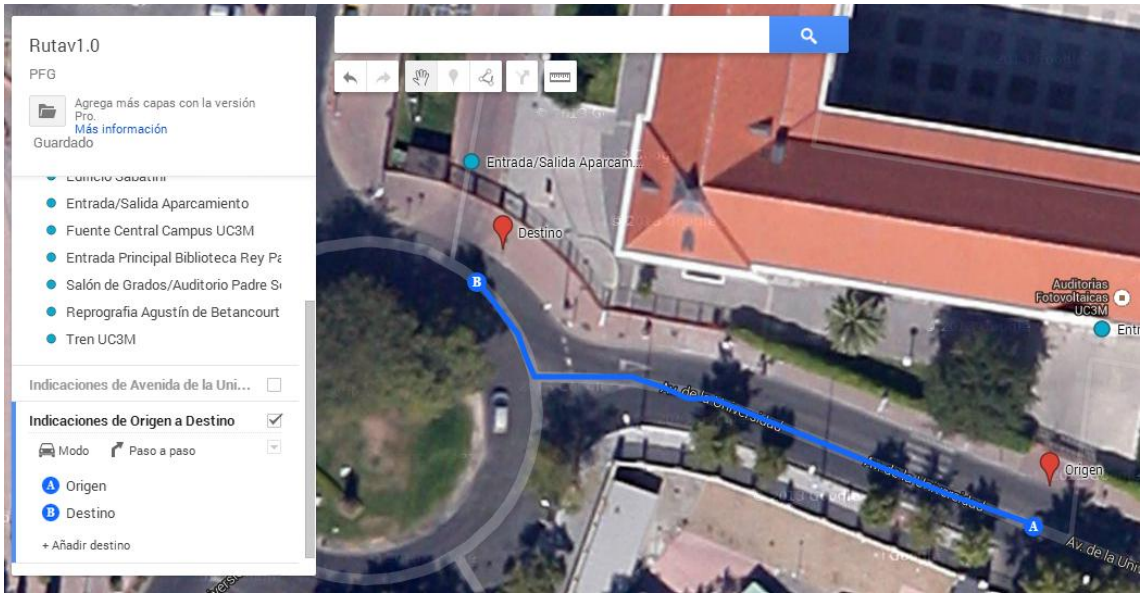


Ilustración 48: Creación ruta (FUENTE: Google)

Como podemos comprobar, la ruta entre ambos puntos será las más corta de manera automática, pero nosotros la modificaremos según a nuestro gusto. Queremos una ruta que pase por todo el campus, rodeando los edificios y que pase cerca de nuestros puntos, para que cuando la simulemos, la realidad aumentada nos muestre los puntos de interés de manera más natural.

Con el cursor, seleccionamos la ruta y podremos desplazarla. La herramienta de Google es capaz de detectar las vías (carreteras, calles peatonales, etc.) por lo que al mover la línea azul esta se adapta a cada calle. En las dos imágenes siguientes se muestra un ejemplo, de cómo podemos crear rutas distintas arrastrando el camino azul.

Ilustración 49: Creación ruta (FUENTE: Google)

Este será el resultado final de nuestra ruta:

Ilustración 50: Creación ruta (FUENTE: Google)

3.2.2. Exportando la ruta

Ahora solo nos falta obtener esto es un archivo que sea legible por un programa. Existen principalmente tres[19]:

- **.kml (Keyhole Markup Languate):** Lenguaje de marcado en XML para representar datos geográficos en 3D
- **.kmz**: Este formato es una variación del KML. Básicamente es una versión comprimida y de menos peso que esta.
- **.gpx (GPS eXchange Format):** Es un método de transferencia de datos GPS en esquema XML.

La principal diferencia entre GPX y KML es la cantidad de datos. El GPX preserva la máxima cantidad de ellos (intensidad de señal GPS, numero de satélites visibles…). En cambio, si solo necesitamos las rutas y nos es indiferente el resto de información, entonces el KML bastará.

[19] Diferencias entre los formatos GPX y KML

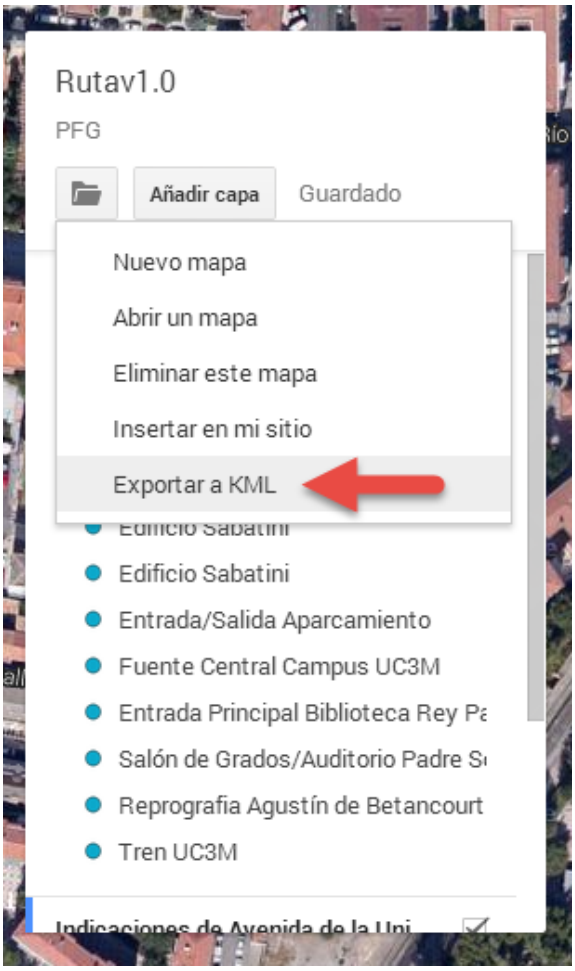


Ilustración 51: Exportación ruta (FUENTE: Google)

3.2.3. Presentado en Google Earth

Google Earth es una extensión de Google Maps; Una variante que nos ofrece información geográfica desde una perspectiva más atractiva, intuitiva y personalizable. Podremos manejar con más facilidad numerosas opciones dirigidas a la medida y a la presentación, como por ejemplo en la creación de videos de visitas turísticas.

A continuación, exportaremos la ruta ya creada con Google Maps. Como era de esperar, Google Earth soporta el formato .KML en su totalidad. Simplemente abriremos el archivo y podremos ver el resultado.

Ilustración 52: Ruta Google Earth (FUENTE: Google Earth)

3.2.4. Simulando la ruta

Una vez tenemos el archivo .KML que nos dice que ruta seguir, ha llegado el momento de simularla. El objetivo es el siguiente: Cargar el archivo en nuestra terminal, y mediante una aplicación ajena, simular una posición ficticia (en este caso una ruta). Con ello "engañaremos" a nuestro móvil acerca de la posición. Y cuando usemos nuestra aplicación de realidad aumentada, los puntos virtuales de interés que se muestren serán los que haya en función de esa posición ficticia.

La aplicación que usaremos será: Simulador rutas GPS (Fernando F. Gallego). La opción que nos permitirá simular nuestro archivo será "Simular KML".

Ilustración 53: Simulación ruta (FUENTE: Simulador de Rutas, Fernando F. Gallego)

En principio nos aparecerá la posición actual. Tendremos un campo donde escribir la dirección de nuestra ruta, y otro donde decir cuánto tiempo queremos tardar en completarla entera.

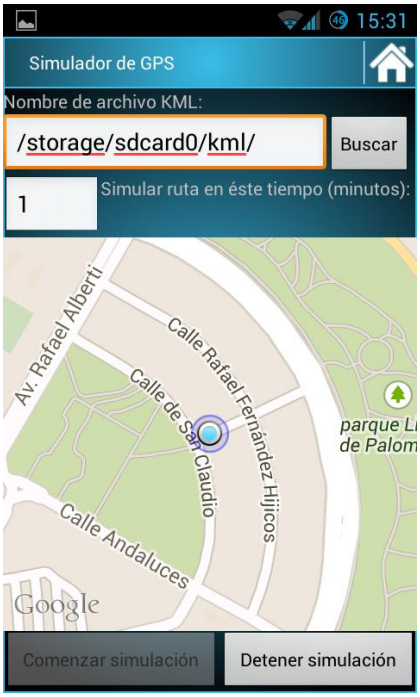


Ilustración 54: Simulación ruta (FUENTE: Simulador de Rutas, Fernando F. Gallego)

Pulsamos sobre “Buscar” y seleccionamos la ruta que previamente hemos exportado al móvil, a la dirección deseada.

Ilustración 55: Simulación ruta (FUENTE: Simulador de Rutas, Fernando F. Gallego)

Ahora tendremos una vista preliminar de la ruta. Solo hará falta darle al botón "Comenzar simulación" para que la aplicación empiece a recorrer la ruta por GPS. Nuestro móvil técnicamente estará circulando a través del camino elegido a ojos del GPS, cuando realmente no nos moveremos.

Ilustración 56: Simulación ruta (FUENTE: Simulador de Rutas, Fernando F. Gallego)

Una vez concluida esta parte, tendremos una aplicación que nos permitirá la prueba de la aplicación final sin salir de casa.

3.3. Desarrollo de un GPS

Será el núcleo de nuestra aplicación. Las librerías de BeyondAR nos ofrecen un sistema de GPS para hacer autosuficiente el marco de trabajo o framework, pero no es tan preciso como lo puede llegar a ser uno provisto por Google Maps.

Tal y como se vio anteriormente, BeyondAR dispone de un sistema de localización que viene de serie, o bien podemos añadir nuestro propio servicio. En este caso, usaremos los Google Services para llevar a cabo esta tarea.

Por lo tanto, antes de empezar a desarrollar la aplicación final de realidad aumentada, crearemos otra aplicación centrada en la localización. Este programa nos dará la latitud, y longitud del terminal en tiempo real, con una precisión y tiempo de actualización bastante alta, además de un mapa provisto por Google Services para mostrar la información.

El nombre será LBS (Location Based Services).

3.3.1. LBSActivity.java

Empezaremos escribiendo el código inicial. El nombre del paquete será descrito como `com.example.aa_lbs`, y a continuación el conjunto de **`imports`** necesarios en el programa (inclusión de librerías).

```
//Nombre del paquete
package com.example.aa_lbs;

//Imports utilizados
//Imports esenciales
import java.util.Calendar;
import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
//Imports referentes al servicio de localizacion
import android.location.Criteria;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
//Imports de elementos del programa (Cuadros de texto, y mensajes de
advertencia).
import android.widget.TextView;
import android.widget.Toast;

//Imports referents a Google Maps y herramientas
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.MapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;
```

Ahora, crearemos la clase principal del programa, que extenderá de la clase Activity y como interfaz usaremos LocationListener, que nos proporcionara métodos imprescindibles para el servicio de localización.

```
//Clase principal
public class LBSActivity extends Activity implements LocationListener
{

…

}
```

Ahora inicializaremos las variables del programa. Entre ellas, las dedicadas a la localización: Con la instancia de LocationManager, oLoc, proporcionamos el servicio que queremos, y con las dos instancias de Location, representaremos las latitudes y longitudes de las dos posiciones (una actual, y otra anterior a ella).

También hay una variable que servirá para mostrar texto (latitud y longitud) y otra para mostrar el mapa y un marcador en él.

```
//Variables
//Instancia del sistema de localización, o del tipo Location que
albergan latitud y longitud

private LocationManager oLoc;
private Location currentLoc;
private Location previousLoc;

TextView txtOutput = null;

private GoogleMap oMap;
Marker oMark = null;
```

Clase **onCreate**. Como hemos visto antes, esta clase estará disponible en toda aplicación, y se ejecutará al iniciarla. En nuestro caso:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      // Asignamos la referencia del layout, en este caso
activity_lbs

      setContentView(R.layout.activity_lbs);

      // Los criterios del GPS (ver punto 2.1.6.3)

      Criteria oGPSSettings = new Criteria();
      oGPSSettings.setAccuracy(Criteria.ACCURACY_FINE);
      oGPSSettings.setSpeedRequired(true);
      oGPSSettings.setAltitudeRequired(true);
      oGPSSettings.setBearingRequired(false);
      oGPSSettings.setCostAllowed(false);
      oGPSSettings.setPowerRequirement(Criteria.POWER_MEDIUM);
```

```
            // Salida de texto par aver latitude y longitude

            txtOutput = (TextView) findViewById(R.id.txtOutput);

            // Asignamos a la instancia oLoc un proveedor del servicio,
      y posteriormente mediante el método getBestProvider, lo
      adaptamos a los criterios anteriormente establecidos.

            oLoc = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);

            String provider = oLoc.getBestProvider(oGPSSettings, true);

            // Condición: Si existe un proveedor y recibimos señal,
      pediremos actualizaciones de posición (cada 1000 milisegundos y
      cada 1 metro). Si no es así, se mostrará un mensaje de error.

            if (provider != null) {
                  oLoc.requestLocationUpdates(provider, 1000, 1, this);
            } else {
                  Toast.makeText(getBaseContext(), "No GPS",
Toast.LENGTH_SHORT)
                              .show();
                  finish();
            }

      }
```

Clase **onLocationChanged**. Implementada en la interfaz de LocationListener. Se ejecutará cada vez que tengamos una nueva localización. En nuestro caso, cada vez que tengamos una nueva longitud y latitud almacenada en una variable tipo Location, mediante los métodos getLatitud y getLongitude mostraremos dichos puntos en la salida de texto.

```
      @Override
      public void onLocationChanged(Location location) {

            // Almacenados la nueva posicion en currentLoc
            currentLoc = location;

            if (previousLoc != null) {

                  String sText = "Lon: "
                  + currentLoc.getLongitude() + "\n"
                  + "Lat: " + currentLoc.getLatitude()
                  + "\n" + "Alt: "
                  + currentLoc.getAltitude();

                  txtOutput.setText(sText);
            }

            // Ahora la posición actual pasa a ser la anterior,
      hacienda sitio a la nueva que vendrá mas adelante.

            previousLoc = currentLoc;
```

```
            // Con el objetivo de mostrar el punto en el mapa, lo
      deberemos hacer creando una nueva variable de tipo LatLng, en la
      cual introducimos la latitude y longitud que hemos recibido.

            LatLng oPos = new LatLng(currentLoc.getLatitude(),
                        currentLoc.getLongitude());

            // En la instancia del mapa, ponemos la referencia, map,
      para posicionarlo en la aplicación.

            oMap = ((MapFragment)
getFragmentManager().findFragmentById(R.id.map))
                        .getMap();

            // Si el mapa existe, deberemos borrar todo lo que
      contiene. De otra manera, cada vez que actualicemos posición y
      la plasmemos en el mapa, todos los marcadores se guardaran y
      quedará una fila de estos. Nosotros solo queremos un marcador
      que nos diga nuestra posición.

            if (oMap != null) {
                  oMap.clear();

            // Añadimos el marcador en oPos, variable creada
      anteriormente, y añadimos un título.

                  oMark = oMap.addMarker(new
MarkerOptions().position(oPos).title(
                              "My Location"));

            // Movemos la camara hacia la posicion oPos con su
      respective zoom.

      oMap.moveCamera(CameraUpdateFactory.newLatLngZoom(oPos, 17));

            }

      }
```

Clase **onProviderDisabled**. Se ejecuta cuando finalice la conexión con el proveedor. En nuestro programa, si eso ocurre, eliminaremos todo contenido de la variable oLoc, y mostraremos un mensaje de error.

```
      @Override
      public void onProviderDisabled(String provider) {
            oLoc.removeUpdates(this);
            Toast.makeText(getBaseContext(), provider + " is
disabled.",
                        Toast.LENGTH_SHORT).show();
      }
```

Clase **onProviderEnabled**. Al contrario que la anterior, se ejecuta si la conexion con el proveedor se reanuda. Eliminaremos el contenido de oLoc nuevamente, pero esta vez solicitaremos al servicio actualizaciones cada 1000 milisegundos o cada 1 metro, además de mostrar el mensaje pertinente.

```
	@Override
	public void onProviderEnabled(String provider) {
		oLoc.removeUpdates(this);
		oLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER,
1000, 1, this);
		Toast.makeText(getBaseContext(), provider + " is enabled.",
				Toast.LENGTH_SHORT).show();

	}
```

Clase **onStatusChanged**. Ejecutado si el proveedor cambia, cuando por ejemplo, se pierde la señal del GPS. En este caso haremos un reinicio, borrando contenido de oLoc y solicitando actualizaciones nuevamente.

```
	@Override
	public void onStatusChanged(String provider, int status, Bundle
extras) {
		oLoc.removeUpdates(this);
		oLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER,
1000, 1, this);
	}
```

Clase **onPause**. Cuando salimos de la aplicación, pararemos de solicitar actualizaciones, con el principal objetivo de ahorrar batería. De esta manera si no la usamos, no haremos uso de la señal.

```
	@Override
	protected void onPause() {
		super.onPause();
		oLoc.removeUpdates(this);
	}
```

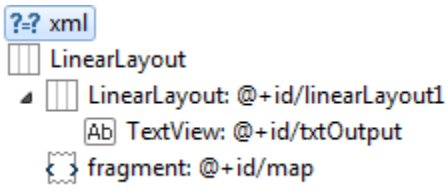
Clase **onResume**. Como en toda aplicación, se ejecuta al volver a usar la actividad una vez se ha pausado.

```
	@Override
	protected void onResume() {
		super.onResume();
		oLoc.removeUpdates(this);
		oLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER,
1000, 1, this);

	}
```

3.3.2. Layout activity_lbs.xml

El layout sobre el que mostraremos la aplicación será muy sencillo. Constará de una salida de texto para mostrar latitud y longitud, y un espacio para el mapa.




```
<!—Se puede observar según la imagen de arriba, que todo se almacena
en un LinearLayout, y dentro de él tenemos un TextView, cuyo id es
txtOutput y mostrará la longitud/latitud según el programa, y un
fragmento, de id map, que muestra el mapa.-->

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <LinearLayout
        android:id="@+id/linearLayout1"
        android:layout_width="fill_parent"
        android:layout_height="105dp"
        android:orientation="horizontal" >

        <TextView
            android:id="@+id/txtOutput"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:textAppearance="?android:attr/textAppearanceSmall"
/>
    </LinearLayout>

<!—Para mostrar el mapa, march_parent ajustará el tamaño de este, y la
clase nos dirigirá al mapa provisto por la librería.-->

    <fragment
        android:id="@+id/map"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        class="com.google.android.gms.maps.MapFragment" />

</LinearLayout>
```

3.3.3. Liberia google-play-services y AndroidManifest.xml

Tal y como se explicaba en el punto 2.1.7, añadiremos la librería en nuestro proyecto. Y para finalizar, nuestro último paso se dará en el archivo llamado AndroidManifest.xml.

En este archivo, se guardaran todas las opciones y configuraciones de nuestro aplicación, como los permisos necesarios para el programa, o la API Key, también vista en el 2.1.7.

```
<!—Encabezado, nombre del paquete y versión del programa-->
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.aa_lbs"
    android:versionCode="1"
    android:versionName="1.0" >

<!—SDK utilizado. Version minima y version recomendada.-->
    <uses-sdk
        android:minSdkVersion="11"
        android:targetSdkVersion="17" />

<!—Permisos (internet, escritura, acceso a la red, acceso a la
posición-->
    <permission
        android:name="com.example.aa_lbs.permission.MAPS_RECEIVE"
        android:protectionLevel="signature" />

    <uses-permission
android:name="com.example.aa_lbs.permission.MAPS_RECEIVE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVIC
ES" />
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
      <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>

<!—Con esto, nos aseguraremos de que soporta OpenGL ES V2-->

    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true" />

<!—Parametros básicos. Nombre, estilo e icono del programa.-->

    <application

        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

<!—API KEY recibida por Google (punto 2.1.7)-->

        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="AIzaSyA4mIL53AQDeKXcXox9mGWgfqAD9Yec03Q" />

        <meta-data
         android:name="com.google.android.gms.version"
         android:value="@integer/google_play_services_version" >
</meta-data>
```

```
        <activity
            android:name="com.example.aa_lbs.LBSActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```

3.3.4. Ejemplo

En la siguiente captura podemos ver la aplicación en funcionamiento. Mientras el mapa muestra nuestro movimiento, los datos en la parte superior nos ofrecen la información al respecto.

Ilustración 57: Aplicación LBS en funcionamiento (FUENTE: Android)

3.4. Desarrollo de la aplicación

Finalmente, en este apartado desarrollaremos la aplicación final y todas sus clases, así como la inclusión del sistema de GPS.

3.4.1. Estructura del proyecto

Como todo proyecto Android, este consta de una jerarquía propia. Las carpetas que más nos interesan están recuadradas en rojo, y en ellas nos centraremos a continuación.

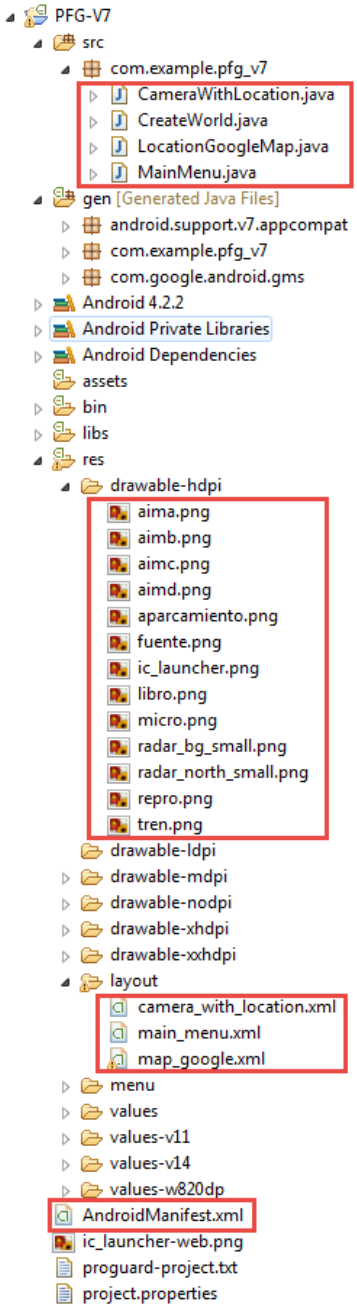


Ilustración 58: Jerarquía Proyecto (FUENTE: Eclipse)

En la carpeta *src*, dentro del paquete *com.example.pfg_v7* encontramos todos los archivos *.java* que compondrán nuestra aplicación.

Dentro de *drawable-hdpi* tendremos todos los recursos de imágenes, como el diseño de los puntos de interés, el radar, etc.

En la carpeta *layout*, los archivos *.xml* que darán apariencia y utilidad física a la aplicación.

Y para finalizar, *AndroidManifest.xml* que resumirá todos los detalles, restricciones, permisos, etc., de nuestro programa.

3.4.1.1. MainMenu.java

Clase encargada de mostrarnos el menú principal para un fácil acceso al programa.

```
/*
*
*VERSION 7.0
*
*Aplicación funcional.
*Altura de los objetos añadida.
*
*
*
*/

//Imports para otorgar las clases, funciones y librerías necesarias.


package com.example.pfg_v7;

import android.os.Bundle;

import com.example.pfg_v7.CameraWithLocation;
import com.example.pfg_v7.CreateWorld;
import com.example.pfg_v7.R;

import android.app.Activity;
import android.app.ListActivity;
import android.content.Intent;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;

//Extendemos de ListActivity para heredar la lista y sus funciones

public class MainMenu extends ListActivity {

	//Valores de la lista, que en este caso solo será uno

	private String[] values = new String[] { "GeoLocalicazion (GPS
independiente)" };

	//Metodo onCreate inicializado al empezar el programa
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		//Referencia del layout
		setContentView(R.layout.main_menu);

		//Ponemos los valores de la lista

		setListAdapter(new ArrayAdapter<String>(this,
				android.R.layout.simple_list_item_1, values));

	}
```

```
	//Listener encargado de ejecutarse cuando pulsamos sobre una opción de la lista. Si pulsamos la posición 0, es decir, la primera opción, ejecutamos el método establecido como openActivity, introduciendo como valor la clase que queremos abrir.

	public void onListItemClick(ListView parent, View v, int position, long id) {

		switch (position) {

		case 0:
			openActivity(CameraWithLocation.class);
			break;
		}

	}

	//Función para abrir actividad. Creamos el intent para pasar la información a la actividad.

	private void openActivity(Class<? extends Activity> ActivityClass) {
		Intent intent = null;
		intent = new Intent(this, ActivityClass);
		startActivity(intent);
	}

	//Cada vez que hagamos abramos la aplicación vaciamos el mundo.

	@Override
	protected void onResume() {
		super.onResume();
		CreateWorld.sharedWorld = null;
	}

}
```

3.4.1.2. CreateWorld.java

Clase encargada de "dibujar" o de crear el mundo virtual que se mostrará en pantalla.

```
/*
 * Creador de mundos
 *
 */

package com.example.pfg_v7;

import android.annotation.SuppressLint;
import android.content.Context;

import com.beyondar.android.world.GeoObject;
import com.beyondar.android.world.World;


@SuppressLint("SdCardPath")
//Clase encargada de crear el mundo.

public class CreateWorld  {
```

```
	//Variable constante
	public static final int LIST_TYPE_EXAMPLE_1 = 1;

	//Instancia de la clase MUNDO.
	public static World sharedWorld;

	//Funcion para generar objetos virtuales en el mundo, que
devuelve un valor de tipo MUNDO.
	public static World generateObjects(Context context) {

		//Si ya existe, lo devolvemos.
		if (sharedWorld != null) {
			return sharedWorld;
		}

		sharedWorld = new World(context);

		//Imagen por defecto si no obtenemos la fuente de la
nuestra (en caso de obtenerla por red).

	sharedWorld.setDefaultImage(R.drawable.beyondar_default_unknow_i
con);

		//Posicion del usuario. Se vera actualizada con el GPS.
		sharedWorld.setGeoPosition(40.390928, -3.635711);

		//Creamos los objetos virtuales.
		GeoObject go1 = new GeoObject(1l); // Constructor
		go1.setGeoPosition(40.390691, 3.636787); //Posicion (lat,
long, altura)
		go1.setImageResource(R.drawable.rectangle); //Imagen
		go1.setName("Ejemplo"); //Nombre

		GeoObject go2 = new GeoObject(2l);
		go2.setGeoPosition(40.331758 ,-3.767044, 0.00025);
		go2.setImageResource(R.drawable.aima);
		go2.setName("Cara A");

		GeoObject go3 = new GeoObject(3l);
		go3.setGeoPosition(40.332114 ,-3.766442, 0.00025);
		go3.setImageResource(R.drawable.aimb);
		go3.setName("Cara B");

		GeoObject go4 = new GeoObject(4l);
		go4.setGeoPosition(40.332592 ,-3.766878, 0.00025);
		go4.setImageResource(R.drawable.aimc);
		go4.setName("Cara C");

		GeoObject go5 = new GeoObject(5l);
		go5.setGeoPosition(40.332239 ,-3.767472, 0.00025);
		go5.setImageResource(R.drawable.aimd);
		go5.setName("Cara D");

		GeoObject go6 = new GeoObject(6l);
		go6.setGeoPosition(40.332017 ,-3.765967);
		go6.setImageResource(R.drawable.fuente);
		go6.setName("Fuente");

		GeoObject go7 = new GeoObject(7l);
		go7.setGeoPosition(40.332044 ,-3.765175);
```

```
            go7.setImageResource(R.drawable.libro);
            go7.setName("Biblioteca");

            GeoObject go8 = new GeoObject(81);
            go8.setGeoPosition(40.332731 ,-3.764894);
            go8.setImageResource(R.drawable.micro);
            go8.setName("Salon de Grados");

            GeoObject go9 = new GeoObject(91);
            go9.setGeoPosition(40.332833 ,-3.765992);
            go9.setImageResource(R.drawable.repro);
            go9.setName("Reprografia");

            GeoObject go10 = new GeoObject(101);
            go10.setGeoPosition(40.332714 ,-3.767214);
            go10.setImageResource(R.drawable.tren);
            go10.setName("Tren");

            GeoObject go11 = new GeoObject(111);
            go11.setGeoPosition(40.331903, -3.767789);
            go11.setImageResource(R.drawable.aparcamiento);
            go11.setName("Aparcamiento");


            //Añadimos los objetos virtuales al mundo.
            sharedWorld.addBeyondarObject(go1);
            sharedWorld.addBeyondarObject(go2);
            sharedWorld.addBeyondarObject(go3);
            sharedWorld.addBeyondarObject(go4);
            sharedWorld.addBeyondarObject(go6);
            sharedWorld.addBeyondarObject(go7);
            sharedWorld.addBeyondarObject(go8);
            sharedWorld.addBeyondarObject(go9);
            sharedWorld.addBeyondarObject(go10);
            sharedWorld.addBeyondarObject(go11);


            //Devolvemos el mundo.
            return sharedWorld;
        }

}
```

3.4.1.3. LocationGoogleMap.java

Clase encargada de mostrar los objetos virtuales en el mapa, además de registrar nuestra posición vía GPS.

```
/*
 * GOOGLE MAPS
 *
 * Clase encargada de obtener nuestra posicion, y mostrarla en un
mapa.
 * Se actualiza con el movimiento.
 * Precision mejorada (gps mejorado)
 * Objetos virtuales funcionando.
 */
```

```
package com.example.pfg_v7;

import android.content.Context;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationManager;
import android.location.LocationListener;
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

import com.beyondar.android.plugin.googlemap.GoogleMapWorldPlugin;
import com.beyondar.android.world.GeoObject;
import com.beyondar.android.world.World;
import com.example.pfg_v7.R;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.GoogleMap.OnMarkerClickListener;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;

//Implementamos las interfaces de los marcadores del mapa y el
Location Listener para el GPS
public class LocationGoogleMap extends FragmentActivity implements
            OnMarkerClickListener, OnClickListener, LocationListener {

      private GoogleMap mMap; //Instancia (MAPA)
      private GoogleMapWorldPlugin mGoogleMapPlugin; //Plugin para
poner los objetos virtuales
      private World mWorld; //Instancia WORLD
      Marker mMark; //Marcador de posicion

      private LocationManager oLoc; //Instancia para el sistema de
localizacion


      @Override
      protected void onCreate(Bundle savedInstanceState) {

            super.onCreate(savedInstanceState);
            setContentView(R.layout.map_google);

            //Referencia del botón en el layout haciendolo visible,
además del Listener.
            Button myLocationButton = (Button)
findViewById(R.id.myLocationButton);
            myLocationButton.setVisibility(View.VISIBLE);
            myLocationButton.setOnClickListener(this);

            //Criterio de nuestro GPS
            Criteria oGPSSettings = new Criteria();
            oGPSSettings.setAccuracy(Criteria.ACCURACY_FINE);
            oGPSSettings.setSpeedRequired(true);
            oGPSSettings.setAltitudeRequired(true);
            oGPSSettings.setBearingRequired(true);
```

```
            oGPSSettings.setCostAllowed(false);
            oGPSSettings.setPowerRequirement(Criteria.POWER_MEDIUM);

            oLoc = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);

            //Añadimos el criterio a nuestro sistema de localizacion
            String provider = oLoc.getBestProvider(oGPSSettings, true);

            //Si existe proveedor y cogemos señal, actualizar cada 1000
milisegundos o 1 metro. Si no, mostrar mensaje.
            if (provider != null) {
                  oLoc.requestLocationUpdates(provider, 1000, 1, this);
            } else {
                  Toast.makeText(getBaseContext(), "No GPS",
Toast.LENGTH_SHORT)
                              .show();
                  finish();
            }

            //Referencia de mapa.
            mMap = ((SupportMapFragment) getSupportFragmentManager()
                        .findFragmentById(R.id.map)).getMap();
            if (mMap == null) {
                  return;
            }

            //Llenamos el mundo por medio de CreateWorld.
            mWorld = CreateWorld.generateObjects(this);

            // Constructor plugin para poner los objetos virtuales en
un mapa
            mGoogleMapPlugin = new GoogleMapWorldPlugin(this);
            // Metemos el mapa en ese plugin
            mGoogleMapPlugin.setGoogleMap(mMap);
            // Con el plugin creado, lo metemos en la instancia de
World (hacer siempre antes de meter objetos virtuales)
            mWorld.addPlugin(mGoogleMapPlugin);

            //Listener para el marcador.
            mMap.setOnMarkerClickListener(this);


      mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(mGoogleMapPlug
in.getLatLng(), 15));

      mMap.animateCamera(CameraUpdateFactory.zoomTo(19), 2000, null);

            //DEFINIMOS POSICION DEL USUARIO.
            mWorld.setGeoPosition(mWorld.getLatitude(),
mWorld.getLongitude());


      }

      //Funcion del LocationListener ejecutada al recibir nueva
posición
      @Override
      public void onLocationChanged(Location location) {
```

```
            //Variable utilizada para poner marcador con addMarker
            LatLng oPos = new LatLng(location.getLatitude(),
location.getLongitude());

            //Referencia
            mMap = ((SupportMapFragment) getSupportFragmentManager()
                        .findFragmentById(R.id.map)).getMap();


            //Si existe el mapa, lo borramos todo y ponemos de nuevo el
marcador en la localización nueva.
            //Lo hacemos para que cada vez que se actualice, no quede
una ristra de marcadores.
            if (mMap != null) {

                  mMap.clear();


                  mMark = mMap.addMarker(new
MarkerOptions().position(oPos).title(
                              "My Location"));

            }

            // Creamos de nuevo los objetos virtuales, ya que se
borraron.
            mWorld = CreateWorld.generateObjects(this);

            //DEFINIMOS DE NUEVO LA POSICION DEL USUARIO.
            mWorld.setGeoPosition(location.getLatitude(),
location.getLongitude());


            //Igual que antes, creamos y cargamos el plugin
            // Constructor plugin para poner los objetos virtuales en
un mapa
            mGoogleMapPlugin = new GoogleMapWorldPlugin(this);
            // Metemos el mapa en ese plugin
            mGoogleMapPlugin.setGoogleMap(mMap);
            // Con el plugin creado, lo metemos en la intancia de World
(hacer siempre antes de meter objetos virtuales)
            mWorld.addPlugin(mGoogleMapPlugin);

      }

      //Listener al pulsar marcador.
      @Override
      public boolean onMarkerClick(Marker marker) {
            // Si pulsamos sobre el objeto virtual sale el mensaje
diciendo el nombre
            GeoObject geoObject =
mGoogleMapPlugin.getGeoObjectOwner(marker);
            if (geoObject != null) {
                  Toast.makeText(this,"Click on a marker owned by a
GeoOject with the name: " + geoObject.getName(),
Toast.LENGTH_SHORT).show();
            }

            return false;
      }
```

```
	//Que actualice de nuevo al volver a iniciar la aplicación
	@Override
	protected void onResume() {
		super.onResume();
		oLoc.removeUpdates(this);
		oLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER,
1000, 1, this);
	}

	//Cuando salgamos, que pare de actualizar para ahorrar bateria
	@Override
	protected void onPause() {
		super.onPause();
		oLoc.removeUpdates(this);
	}

	//Al pulsar vamos a la localización del usuario en el mapa.
	@Override
	public void onClick(View v) {

		LatLng userLocation = new LatLng(mWorld.getLatitude(),
mWorld.getLongitude());


	mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(userLocation,
15));
		mMap.animateCamera(CameraUpdateFactory.zoomTo(19), 2000,
null);
	}

	//Si perdemos la señal GPS  por ejemplo, volvemos a reiniciar
las actualizaciones
	@Override
	public void onStatusChanged(String provider, int status, Bundle
extras) {
		oLoc.removeUpdates(this);
		oLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER,
1000, 1, this);

	}

	//Si activamos el proveedor, actualizamos y mostramos el mensaje
	@Override
	public void onProviderEnabled(String provider) {
		oLoc.removeUpdates(this);
		oLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER,
1000, 1, this);
		Toast.makeText(getBaseContext(), provider + " is enabled.",
				Toast.LENGTH_SHORT).show();
	}

	//Si desactivamos el proveedor, las actualizaciones finalizan y
mostramos el mensaje
	@Override
	public void onProviderDisabled(String provider) {
		oLoc.removeUpdates(this);
		Toast.makeText(getBaseContext(), provider + " is
disabled.",
				Toast.LENGTH_SHORT).show();
```

```
	}

}
```

3.4.1.4. CameraWithLocation.java

Esta clase se encarga de la muestra de objetos virtuales en tiempo real mediante la cámara, así como la actualización de la posición del usuario a medida que se mueve, y la de los objetos virtuales, que cambian de tamaño en función de la distancia a la que estas.

```
/*
 * CAMARA (mostrar objetos virtuales)
 *
 * Mejorar sensibilidad y respuesta
 * Mejorar respuesta radar
 */

package com.example.pfg_v7;

import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.Window;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;
import android.widget.Toast;

import com.beyondar.android.fragment.BeyondarFragmentSupport;
import com.beyondar.android.plugin.radar.RadarView;
import com.beyondar.android.plugin.radar.RadarWorldPlugin;
import com.beyondar.android.world.World;

//Clase principal, que extiende de la clase FragmentActivity para
implementar la cámara, y usa las interfaces correspondientes para el
uso de las herramientas y métodos.

public class CameraWithLocation extends FragmentActivity implements
            OnSeekBarChangeListener, OnClickListener, LocationListener
{

	private BeyondarFragmentSupport mBeyondarFragment;
	private World mWorld;

	// Exactamente igual que en la clase LocationGoogleMap.java,
creamos el GPS.

	private LocationManager oLoc;

	//Barra renderizado y botón que se dirige al mapa
```

```
	private SeekBar mSeekBarMax;
	private Button mShowMap;

	//Radar que mostrará los objetos cercanos
	private RadarView mRadarView;
	private RadarWorldPlugin mRadarPlugin;

	//Función onCreate llamada cuando la aplicacion es creada
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);

		// Escondemos la ventana para ver en pantalla completa
		requestWindowFeature(Window.FEATURE_NO_TITLE);

		//Llamamos la function para cargar el layout
		loadViewFromXML();

		// Al igual que en la clase LocationGoogleMap.java, creamos
el GPS.
		Criteria oGPSSettings = new Criteria();
		oGPSSettings.setAccuracy(Criteria.ACCURACY_FINE);
		oGPSSettings.setSpeedRequired(true);
		oGPSSettings.setAltitudeRequired(true);
		oGPSSettings.setBearingRequired(true);
		oGPSSettings.setCostAllowed(false);
		oGPSSettings.setPowerRequirement(Criteria.POWER_MEDIUM);

		oLoc = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);

		String provider = oLoc.getBestProvider(oGPSSettings, true);

		if (provider != null) {
			oLoc.requestLocationUpdates(provider, 1000, 1, this);
		} else {
			Toast.makeText(getBaseContext(), "No GPS",
Toast.LENGTH_SHORT)
					.show();
			finish();
		}


		//A partir de aqui, comparado con LocationGoogleMap, el
codigo cambia
		//Llamamos a generateObjetcs del CreateWorld y lo metemos
en mWorld...
		mWorld = CreateWorld.generateObjects(this);
		// ... y se envia al fragment de la camara
		mBeyondarFragment.setWorld(mWorld);

		//El plugin del radar se mete en mWorld
		mWorld.addPlugin(mRadarPlugin);

		// Metodo usado para ver los FPS
		mBeyondarFragment.showFPS(true);

		// DEFINIMOS POSICION USUARIO.
		mWorld.setGeoPosition(mWorld.getLatitude(),
mWorld.getLongitude());
```

```
	}

	//Funcion para cargar los elementos en pantalla y referencias
	private void loadViewFromXML() {

		//Referencia del layout
		setContentView(R.layout.camera_with_location);

		//Referencia del fragmento
		mBeyondarFragment = (BeyondarFragmentSupport)
getSupportFragmentManager()
				.findFragmentById(R.id.beyondarFragment);

		//Referencia de la barra de renderizado, listener y
parametro
		mSeekBarMax = (SeekBar) findViewById(R.id.seekBar1);
		mSeekBarMax.setOnSeekBarChangeListener(this);
		mSeekBarMax.setMax(100);

		//Referencia del boton y listener
		mShowMap = (Button) findViewById(R.id.showMapButton);
		mShowMap.setOnClickListener(this);

		//Referencia del radar
		mRadarView = (RadarView) findViewById(R.id.radarView);

		// Creamos el plugin del radar
		mRadarPlugin = new RadarWorldPlugin(this);

		// Metemos en el plugin dicho radar
		mRadarPlugin.setRadarView(mRadarView);

		// Decimos que distancia maxima es capaz de ver el radar
		mRadarPlugin.setMaxDistance(100);

		// Color de los puntos
		mRadarPlugin.setListColor(CreateWorld.LIST_TYPE_EXAMPLE_1,
				Color.RED);

		// Tamaño del os puntos

	mRadarPlugin.setListDotRadius(CreateWorld.LIST_TYPE_EXAMPLE_1,
3);

	}

	// Al hacer click al boton vamos al mapa mediante un intent
	@Override
	public void onClick(View v) {
		if (v == mShowMap) {
			Intent intent = new Intent(this,
LocationGoogleMap.class);
			startActivity(intent);
		}
	}

	// Controladores de las barras de renderizado.
	@Override
```

```
	public void onProgressChanged(SeekBar seekBar, int progress,
boolean fromUser) {
		if (mRadarPlugin == null)
			return;
		if (seekBar == mSeekBarMax) {
			mBeyondarFragment.setMaxFarDistance(progress);
			mRadarPlugin.setMaxDistance(progress);
		}
	}

	@Override
	public void onStartTrackingTouch(SeekBar seekBar) {
	}

	@Override
	public void onStopTrackingTouch(SeekBar seekBar) {
	}


	// Al cambiar la posición los objetos cambian en la cámara en
directo. Por ello, implementamos el método que se ejecutara al recibir
una nueva posicion.
	@Override
	public void onLocationChanged(Location location) {

		// Llenamos el mundo
		mWorld = CreateWorld.generateObjects(this);

		// Ponemos la posicion del usuario con la nueva
localizacion
		mWorld.setGeoPosition(location.getLatitude(),
location.getLongitude());

		// Una vez creado el mundo y la nueva posicion, se manda al
fragmento. De esta manera los objetos que vemos en la camara se
actualizan y se mueven según nos movemos nosotros.
		mBeyondarFragment.setWorld(mWorld);

		mWorld.addPlugin(mRadarPlugin);

		mBeyondarFragment.showFPS(true);

	}



	//Metodos propios de la interfaz LocationListener, cuyas
funciones son las mismas que en la anterior clase.
	@Override
	public void onStatusChanged(String provider, int status, Bundle
extras) {
		oLoc.removeUpdates(this);
		oLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER,
1000, 1, this);

	}

	@Override
	public void onProviderEnabled(String provider) {
		oLoc.removeUpdates(this);
```

```
            oLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER,
1000, 1, this);
            Toast.makeText(getBaseContext(), provider + " is enabled.",
                        Toast.LENGTH_SHORT).show();
      }

      @Override
      public void onProviderDisabled(String provider) {
            oLoc.removeUpdates(this);
            Toast.makeText(getBaseContext(), provider + " is
disabled.",
                        Toast.LENGTH_SHORT).show();

      }

      @Override
      protected void onResume() {
            super.onResume();
            oLoc.removeUpdates(this);
            oLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER,
1000, 1, this);
      }

      @Override
      protected void onPause() {
            super.onPause();
            oLoc.removeUpdates(this);
      }

}
```

3.4.1.5. main_menu.xml

Muestra el menú principal. Solo consta de una lista.

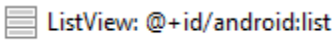


Ilustración 59: main_menu.xml (FUENTE: Eclipse)

```
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="match_parent"
 android:id="@+id/android:list"
 android:layout_height="match_parent" >
</ListView>
```

3.4.1.6. map_google.xml

Layout del mapa. Dentro del Framelayout, es decir, de una capa "marco", colocamos el fragmento del mapa, y un botón con sus respectivos ID's.

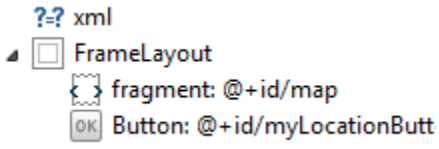


Ilustración 60: map_google.xml (FUENTE: Eclipse)

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <fragment
        android:id="@+id/map"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        class="com.google.android.gms.maps.SupportMapFragment" />

    <Button
        android:id="@+id/myLocationButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="gone"
        android:text="My location" />

</FrameLayout>
```

3.4.1.7. camera_with_location.xml

Layout de la cámara. Dentro del Frame, tendremos un fragmento que será la cámara, y luego dentro de otro fragmento, insertamos desde la librería el radar. Para acabar dentro del primer Frame tenemos una capa llamada RelativeLayout, que dispone ciertos elementos en distintas posiciones, como una salida de texto y la barra de renderizado, o el botón que dirige al mapa.

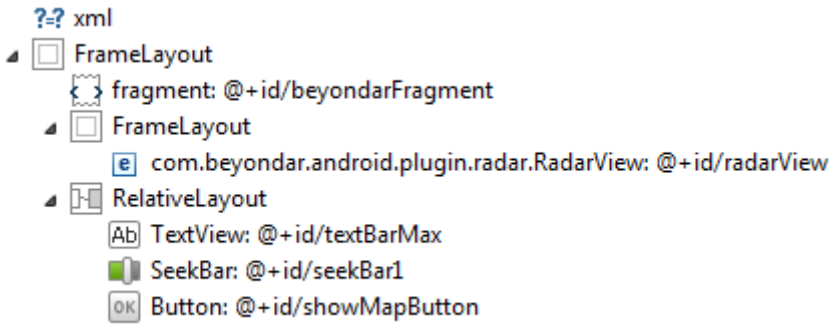


Ilustración 61: camera_with_location.xml (FUENTE: Eclipse)

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <fragment
        android:id="@+id/beyondarFragment"

android:name="com.beyondar.android.fragment.BeyondarFragmentSupport"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <FrameLayout
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_gravity="right|top"
        android:background="@drawable/radar_bg_small" >

        <com.beyondar.android.plugin.radar.RadarView
            android:id="@+id/radarView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/radar_north_small" />
    </FrameLayout>

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        android:orientation="vertical" >

        <TextView
            android:id="@+id/textBarMax"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Max far: " />

        <SeekBar
            android:id="@+id/seekBar1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="2dip"
            android:layout_toLeftOf="@+id/showMapButton"
            android:layout_toRightOf="@id/textBarMax" />

        <Button
            android:id="@+id/showMapButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:layout_alignParentTop="true"
            android:text="Show map" />
    </RelativeLayout>

</FrameLayout>
```

4. Evaluación y resultados

4.1. Evaluación

La evaluación de la aplicación final se basó en la prueba del código con cada cambio que se realizaba, para comprobar que funcionaba acorde a lo establecido. Se iba ejecutando la aplicación en el dispositivo móvil, y mediante la simulación de la ruta, se verificaba que los aspectos añadidos funcionaban.

Con la aplicación ya acabada, se realizaron debugs y pruebas con gente no familiarizada con la aplicación para pudieran ofrecer un feedback de cómo mejorar y arreglar ciertos aspectos de esta.

4.2. Resultados

Cuando abrimos la aplicación, se nos muestra una lista de una sola opción, con el objetivo de que si en un futuro deseemos añadir otras funcionalidades, podamos hacerlo fácilmente. En la opción que tenemos, Geolocalización (GPS independiente) podemos iniciar la actividad.

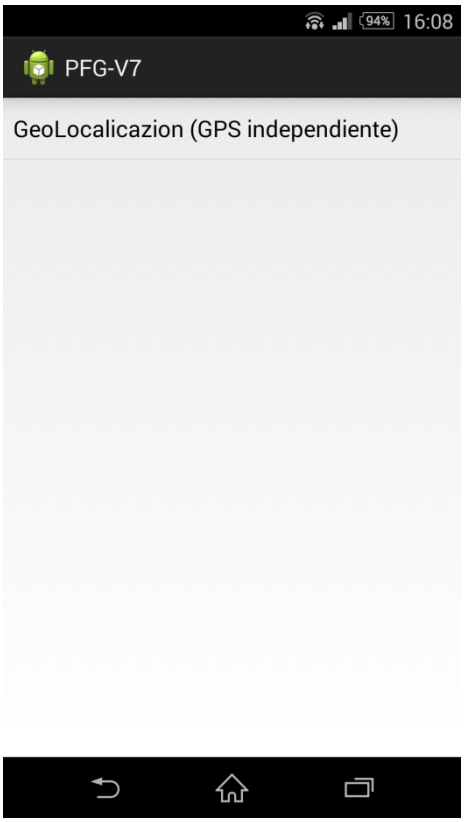


Ilustración 62: Menú principal (FUENTE: Android)

En el momento de la realización de las capturas, no nos situábamos en la Escuela Politecnica, por lo que las imágenes del mundo real no corresponderían, pero gracias a la simulación de

rutas, los objetos virtuales si corresponden a como se verían en la realidad si estuviéramos en la universidad. El punto simulado es el siguiente:

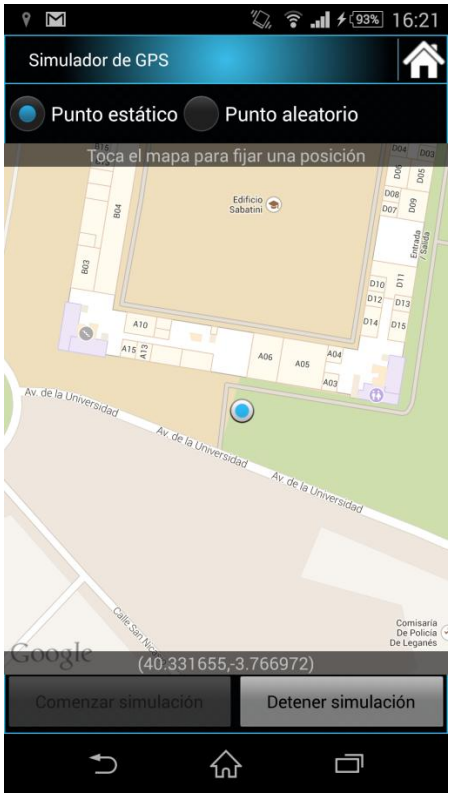


Ilustración 63: Punto geográfico simulado (FUENTE: Android)

En la siguiente imagen, podemos apreciar distintos puntos, como la fuente o la reprografía, mientras que hay otros puntos que se sitúan en altura, como son las caras C y B del Edificio Sabatini.

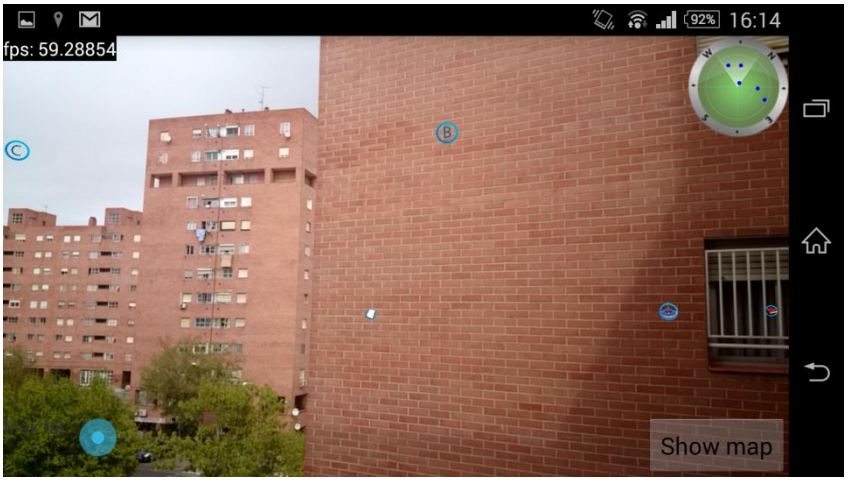


Ilustración 64: Puntos virtuales (FUENTE: Android)

Como curiosidad, si alzamos la vista, encontraremos el punto en altura A, perteneciente a la cara A del Edificio Sabatini, que por ángulo está encima de nuestras cabezas.

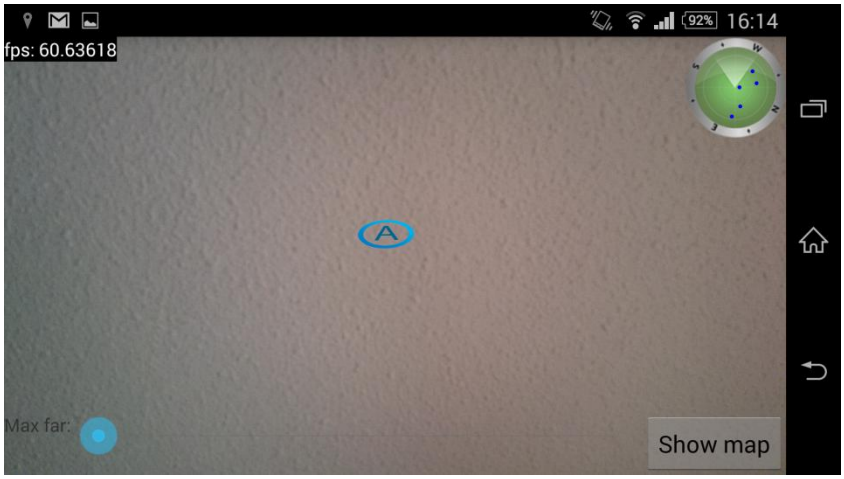


Ilustración 65: Puntos virtuales en altura (FUENTE: Android)

5. Futuras líneas para mejorar el proyecto

Esta aplicación, está dirigida a su uso en dispositivos vestibles, concretamente en gafas de Realidad Aumentada. Nuestro programa no es más que una parte del sistema completo. Introducimos en el código una latitud, una longitud, y una altura, y seremos capaces de verlo en frente de nuestros ojos.

Pero aún queda mucho camino por recorrer. Sobretodo en el aspecto de la accesibilidad. El tener que modificar el código cuando queramos incluir una dirección es un límite de cara al usuario, pero como hemos descrito esto es solo parte del programa completo.

Una línea de futuro será el desarrollo de un programa que reciba la dirección de una calle, y por Geocoding[20], transformarlo en un punto con latitud y longitud. Posteriormente llevaríamos este punto a nuestro programa para situarlo por Realidad Aumentada. Lo mismo para monumentos, edificios, zonas de interés, etc.

Otra línea de futuro seria la adicción y personalización de puntos. Poder añadir puntos favoritos, insertar la imagen que queramos, además del nombre, y luego poder verlo.

[20] Definición de Geocoding

6. Presupuesto

El presente Proyecto Final de Grado se empezó al final de mi estancia Erasmus, 1 de Marzo del 2014, y finaliza el actual mes, a finales de Septiembre del 2014. Hacen un total de 7 meses aproximadamente, todo ello compatibilizado con 2 asignaturas en el 2º cuatrimestre del 4º año, y prácticas en empresa a jornada completa durante todo el curso académico.

A continuación, según las fases del proyecto, haremos una aproximación del número de horas que ha llevado cumplir cada parte, con un mínimo de 2-3 horas diarias.

- **Fase 1: Introducción al lenguaje de programación JAVA:** El mes de Marzo, aproximadamente 90 horas.
- **Fase 2: Adecuación del entorno de desarrollo:** Principios del mes de Abril, 3 horas.
- **Fase 3: Documentación para programación de aplicaciones Android**: Abril aproximadamente 90 horas.
- **Fase 4: Realización de aplicaciones Android:** Durante mitad del mes de Mayo, aproximadamente 60 horas.
- **Fase 5: Desarrollo de fases previas de la Aplicación final; Fase 6: Creación de un GPS; Fase 7: Servicios de Google:** Simultáneamente se hacen los tres procesos a la vez durante el mes de Junio, 90 horas. Dividimos cada proceso en 30 horas.
- **Fase 8: Adecuación de la Aplicación final; Fase 9: Últimos retoques.:** Mes de Julio y principios de Agosto, 40 horas (20 horas cada uno).
- **Fase 10: Testeo y prueba de errores definitiva:** Principio del mes de Septiembre, 10 horas.
- **Fase 11: Redacción de la memoria:** Simultáneamente durante todas las fases, un tiempo aproximado de 100 horas.

En la siguiente tabla 1, se plasma el tiempo aproximado por fase, así como el total de horas invertidas.

Fase 1: Introducción al lenguaje de programación JAVA	90 horas
Fase 2: Adecuación del entorno de desarrollo	3 horas
Fase 3: Documentación para programación de aplicaciones Android	90 horas
Fase 4: Realización de aplicaciones Android	60 horas
Fase 5: Desarrollo de fases previas de la Aplicación final	30 horas
Fase 6: Creación de un GPS	30 horas
Fase 7: Servicios de Google	30 horas
Fase 8: Adecuación de la Aplicación final	20 horas
Fase 9: Últimos retoques	20 horas
Fase 10: Testeo y prueba de errores definitiva	10 horas
Fase 11: Redacción de la memoria	120 horas
Total	**503 horas**

Tabla 2: Fases y horas del proyecto

A continuación, en la tabla 2, se hace un desglose del precio de los materiales y recursos utilizados.

Concepto	Precio	Amortización	Importe
Ordenador de sobremesa Intel® Core ™ i7-2600k CPU @ 3.4GHz - 3.7 GHz, 4GB RAM	999	2/4	499,5
Teléfono móvil Samsung Galaxy S2 i9100 1GB	599	2/4	299,5
Teléfono móvil Sony Xperia Z2 3GB	629	1/4	157,25
Java SE Development Kit 7 Update 51 (64-bit)		-	
Eclipse IDE for Java Developers Version: Kepler Service Release 2		-	
BeyondAR Framework		-	
Cuenta Google Developer Console		-	
Google Maps / Google Earth		-	
Aplicación Simulador de rutas GPS, por Fernando F. Gallego		-	
TOTAL			956,25

Tabla 3: Costes de material

Suponiendo el coste de material anterior, además de los costes de personal (suponiendo 20€/hora), los costes totales son:

Concepto	Importe
Costes de personal	10.060 €
Costes de material	956,25 €
Costes indirectos (20%)	2.203,25 €
Base imponible	13.219,25 €
IVA (21%)	2776,1 €
TOTAL	**15.995,34 €**

7. Conclusiones

7.1. Consecución de objetivos

Con el presente proyecto, nos establecidos una serie de objetivos principales y secundarios, que nos han llevado a su cumplimentación:

Como objetivo principal, hemos desarrollo una aplicación de realidad aumentada, que ha resuelto la propuesta establecida: Ser capaces de encontrar un punto terrestre, es decir, con una latitud y una longitud, mediante el uso de Realidad Aumentada y la cámara. Este punto variará en función de nuestra posición, mediante el GPS que hemos creado..

Como objetivos secundarios, hemos estudiado y aprendido el lenguaje Java, teniendo un primer y exhaustivo contacto con él. Hemos llegado a controlar sus funcionalidades, sintaxis, etc., facilitando un futuro aprendizaje a mayor nivel.

Hemos aprendo a usar como IDE el programa Eclipse y sus diversas funciones, sobretodo en el caso del uso de herramientas Android.

Con las herramientas conseguidas para desarrollar Android, he creado mis primeras aplicaciones funcionales abriendo un mundo de posibilidades de aquí en adelante, gracias a las oportunidades que nos ofrece este sistema operativo.

También hemos tenido un primer contacto con la Realidad Aumentada y aprendido sus bases, usos más importantes y la creación de aplicaciones que hacen uso de esta ciencia. Así como los sistemas de localización, que hasta ahora me eran desconocidos. Pero, gracias a este proyecto, tendremos un conocimiento de ellos mucho mayor.

7.2. Conclusión personal

En la realización de este proyecto hemos tenido que empezar a estudiar desde cero diversas tecnologías y lenguajes, como Java, el entorno Eclipse, Android, etc., por lo que en un principio el esfuerzo tuvo que alcanzar un nivel mayor con el objetivo de aprender a usar herramientas que nunca antes las había visto o desarrollado con ellas.

Aun así, el esfuerzo y las ganas de aprender otras materias han dado su fruto y sin duda el resultado ha merecido la pena, sobre el tiempo invertido y las horas de aprendizaje.

El tiempo sin duda ha sido otra característica a tener en cuenta; Realizando prácticas de empresa al mismo tiempo, me resulto bastante complicado buscar tiempo para la realización del proyecto, pero con una correcta organización finalmente se pudo llevar a cabo.

Como conclusión final, el haber decidido hacer este proyecto ha sido una de las mejores decisiones que he tomado durante mi estancia en la universidad. Me ha permitido aprender elementos que antes desconocía, y que siempre me habían atraído. Ojala estos conocimientos me sean de utilidad en un futuro, y que quizás, algún día, tenga que utilizarlos de nuevo.

Porque sin duda, lo que he adquirido me acompañará durante el resto de mi labor como ingeniero.

Bibliografía

Azuma, R. T. (August 1997). *A Survey of Augmented Reality.* Hughes Research Laboratories.

Comparacion de SDK's para Realidad Aumentada. Obtenido de http://socialcompare.com/en/comparison/augmented-reality-sdks

Fernández Santiago, R., González Gutiérrez, D., & Remis García, S. *Realidad Aumentada.* Escuela Politécnica de Ingeniería de Gijón (E.P.I. Gijón) Universidad de Oviedo.

Garcia de Jalón, J., Ignacio Rodriguez, J., Mingo, I., Imaz, A., Brazález, A., Larzabal, A., y otros. (2000). *Aprenda Java como si estuviera en primero.* San Sebastián: Tecnun.

Gómez Oliver, S. *Curso Programaciñon Android.* www.sgoliver.net.

Google - Android Developer. Obtenido de http://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle

Puig Sanz, J. *BeyondAR*. Obtenido de http://beyondar.github.io/beyondar/doxygen/index.html

Tushinsky, A. Android Complete!

Wikipedia - Android. Obtenido de http://es.wikipedia.org/wiki/Android

ABC - Primer smartphone de la historia. Obtenido de http://www.abc.es/20120222/tecnologia/abci-simon-primer-smartphone-historia-201202221308.html